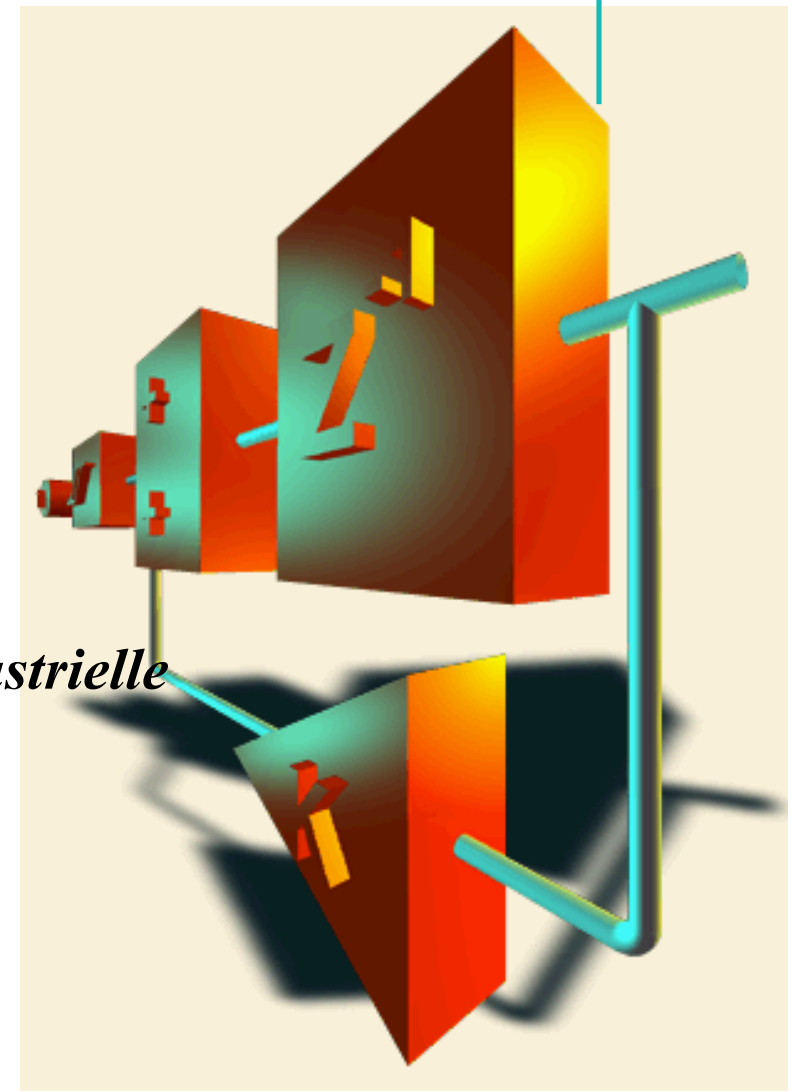
Simulink

Formation Simulink



Pr. Khalid BENJELLOUN

Section Automatique et Informatique Industrielle

Département Electrique

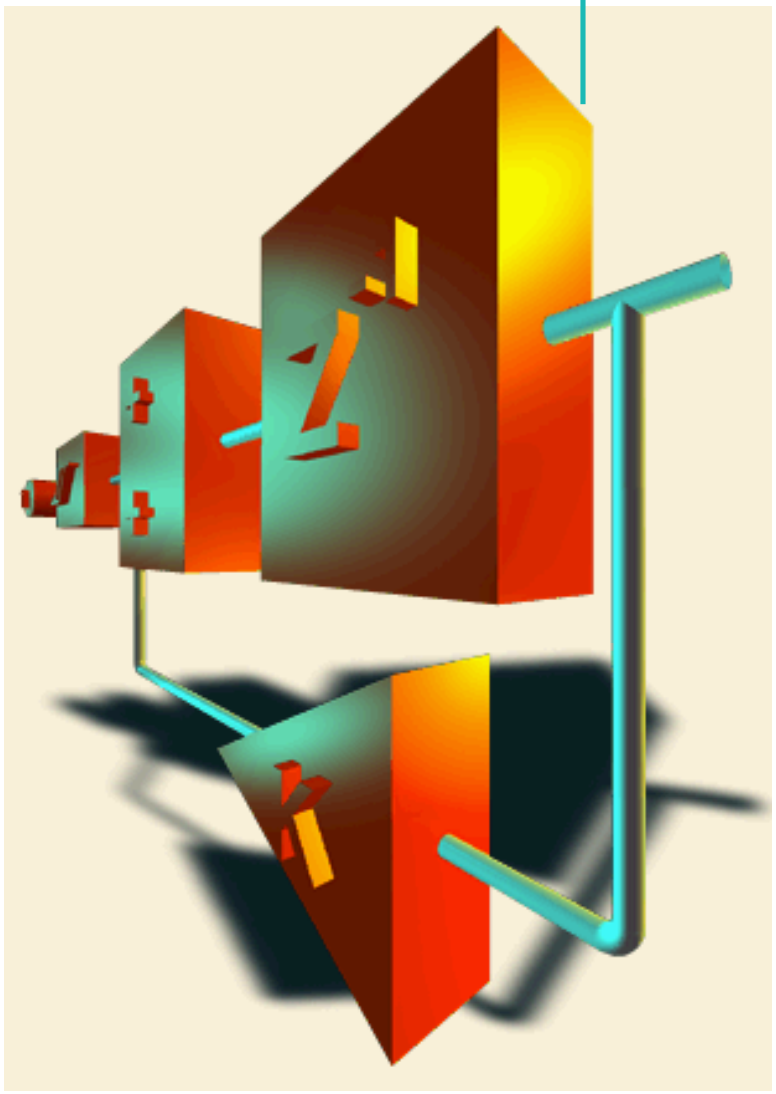
Ecole Mohammadia des Ingénieurs

Sommaire

- Introduction 1
- Les Bibliothèques de Simulink 11
- Construction de systèmes 23
- Simulation 39
- Construction de blocs 65
- Etude à partir de MATLAB 83
- Moteur à Courant Continu 97
- Sous systèmes à exécution conditionelle 115
- Boucles algébriques 125
- S-function 129
- Exercice : rebond d'une balle 153

Simulink

1. Introduction



1.1 Plan du Cours

- **Les Bibliothèques de Simulink**
- **Construction de systèmes**
- **Simulation**
- **Construction de blocs**
- **Etude à partir de MATLAB**
- **Moteur à Courant Continu**
  - Modélisation
  - Régulation
  - Optimisation
- **Sous-Systèmes à exécution conditionnelle**
- **Boucles algébriques**
- **S-Function**
- **Exercice : rebond d'une balle**

1.2 Information de Démarrage

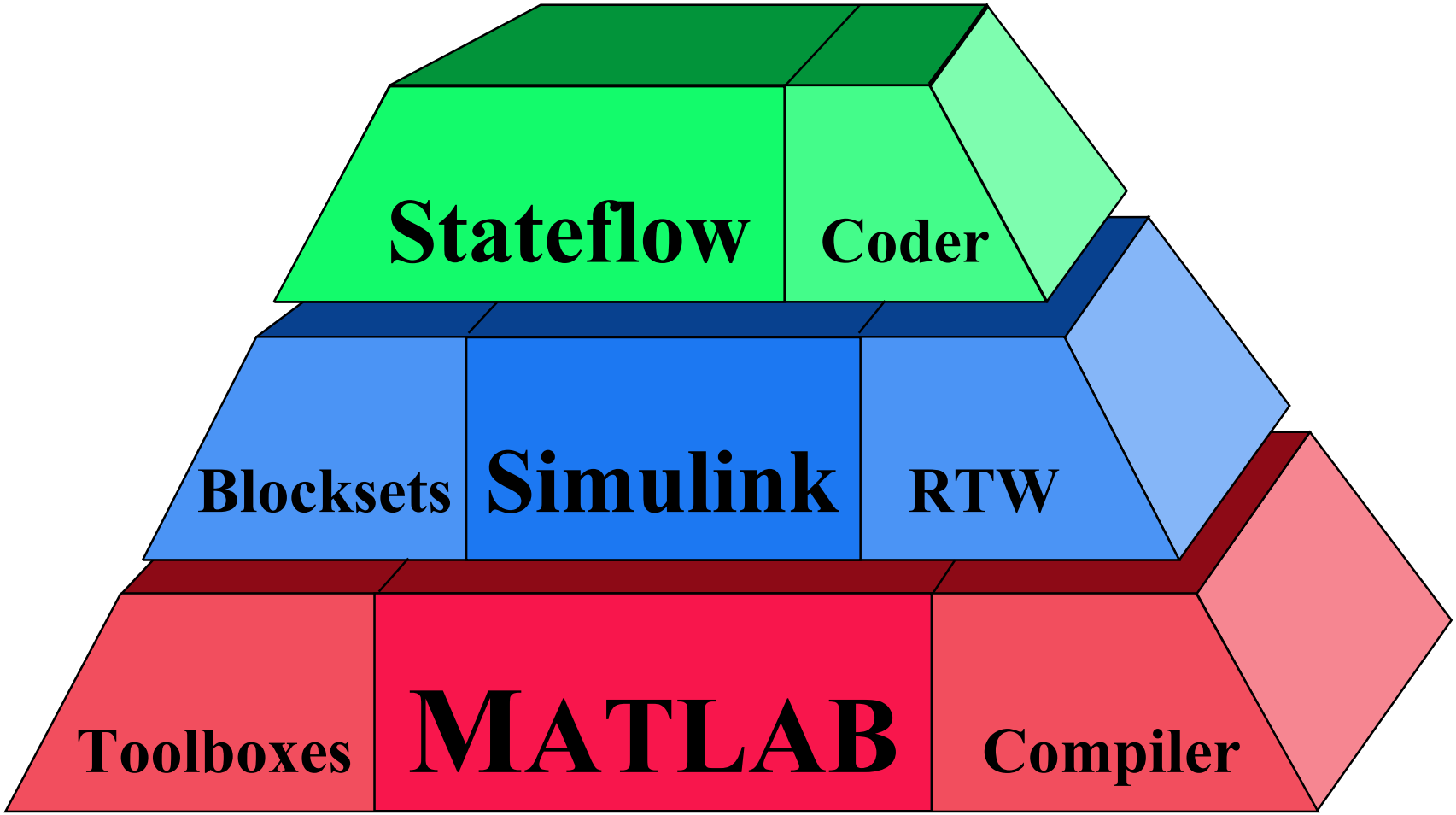
- **Ouvrir MATLAB**

  Double-clic sur l'icone

- **Utiliser Simulink**

  Taper `simulink` dans la fenêtre MATLAB

1.3 Les Produits MATLAB



1.4 Qu'est-ce que Simulink?

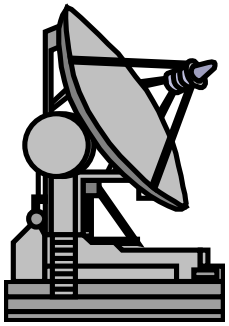
Un outil de Simulation intégré à l'environnement de programmation scientifique MATLAB

Ceci signifie, entre autres :

- Edition de blocs
- Simulation de modèles non-linéaires
- Simulation de modèles discrets ou continus
- Simulation de modèles hybrides
- Simulation asynchrone (échantillonnage non-uniforme)
- Intéraction avec Matlab, ses extensions, ses boîtes à outils

1.5 Que peut-on modéliser avec Simulink?

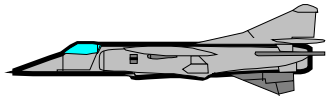
Ce que l'on peut modéliser mathématiquement



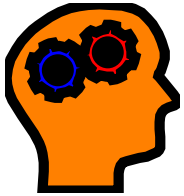
Systèmes de Communications Satellites

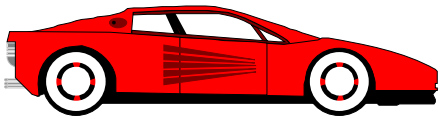
Systèmes de Navigation

Systèmes Aéronautiques



Systèmes Biologiques





Systèmes Automobiles

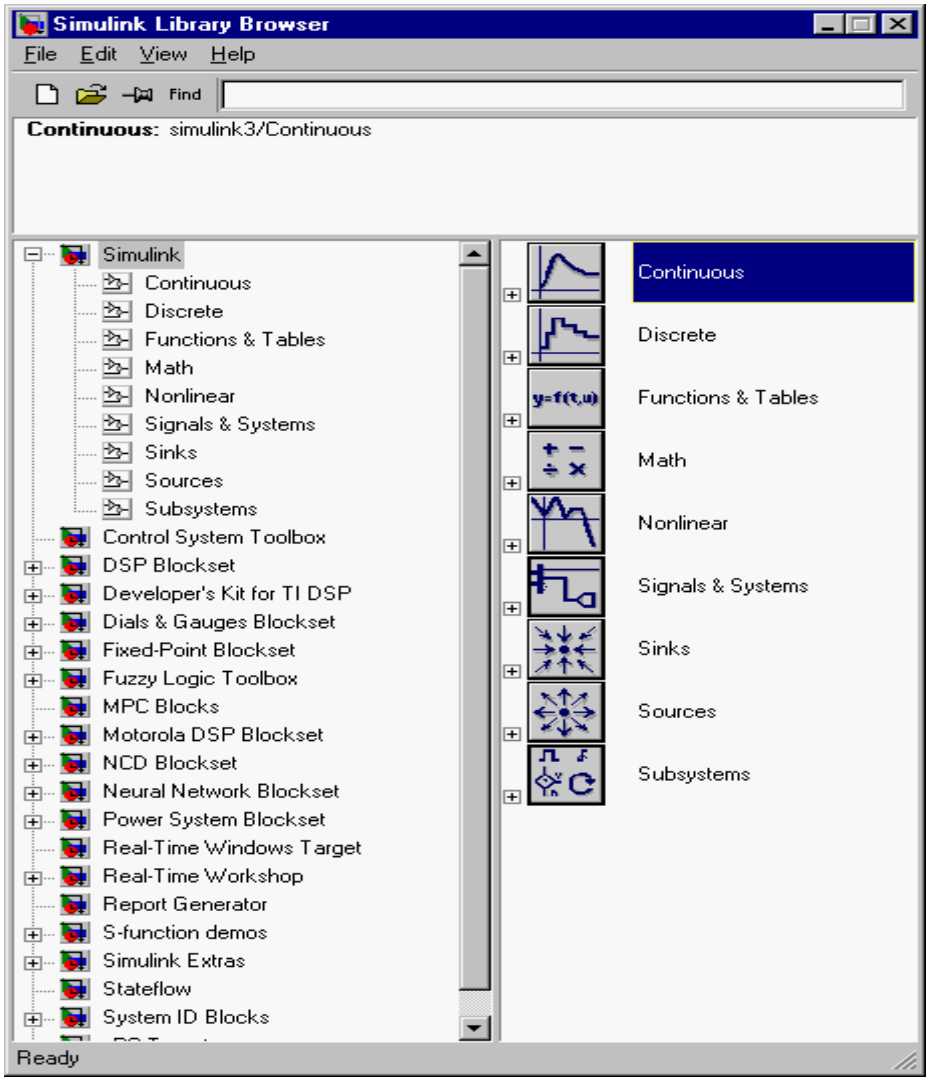
Systèmes Monétaires

1.6 Exécution de Simulink

Taper dans MATLAB la commande :

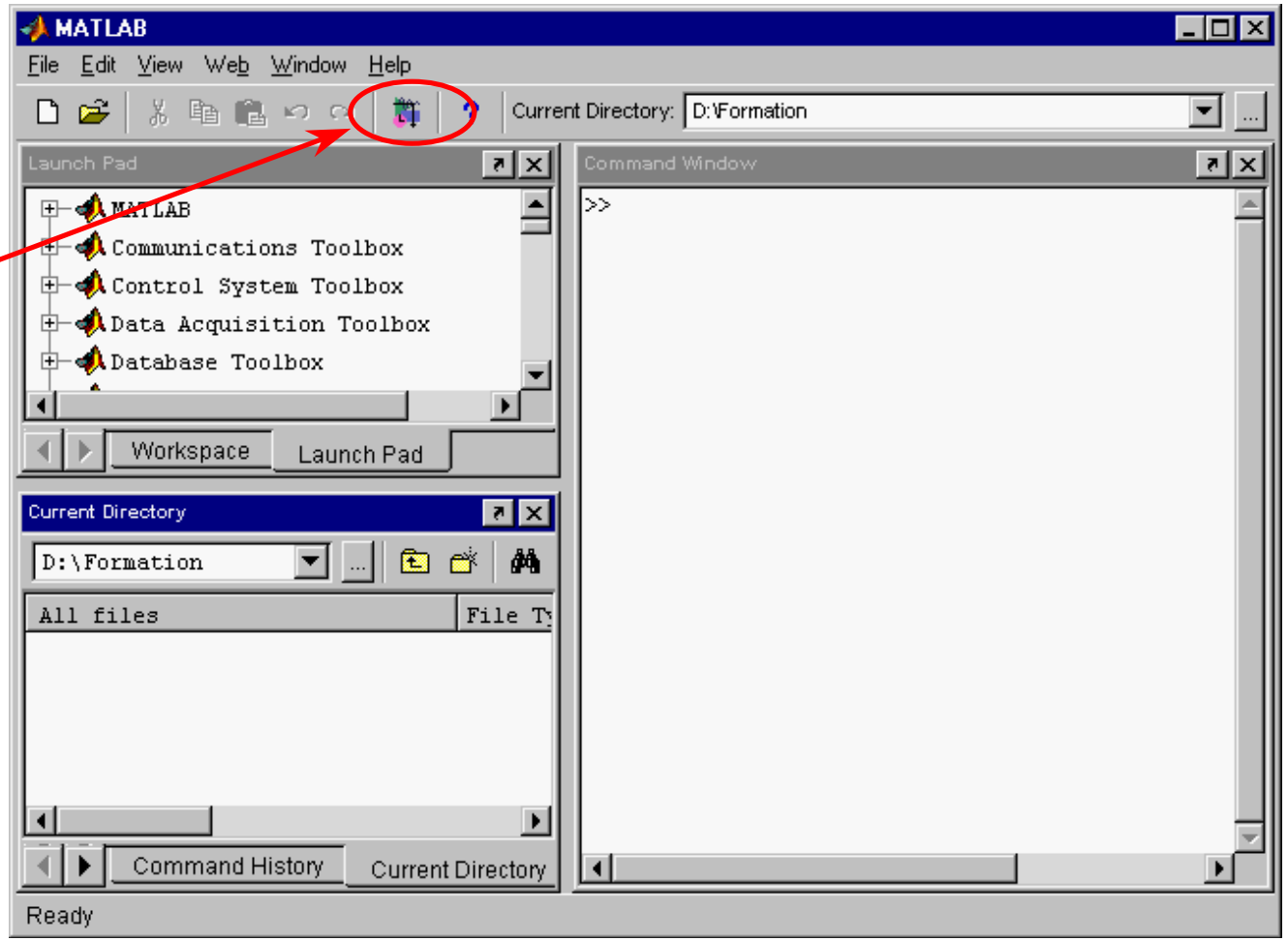
```
>> simulink
```

et le navigateur SIMULINK apparaît.



1.6 Exécution de Simulink (cont.)

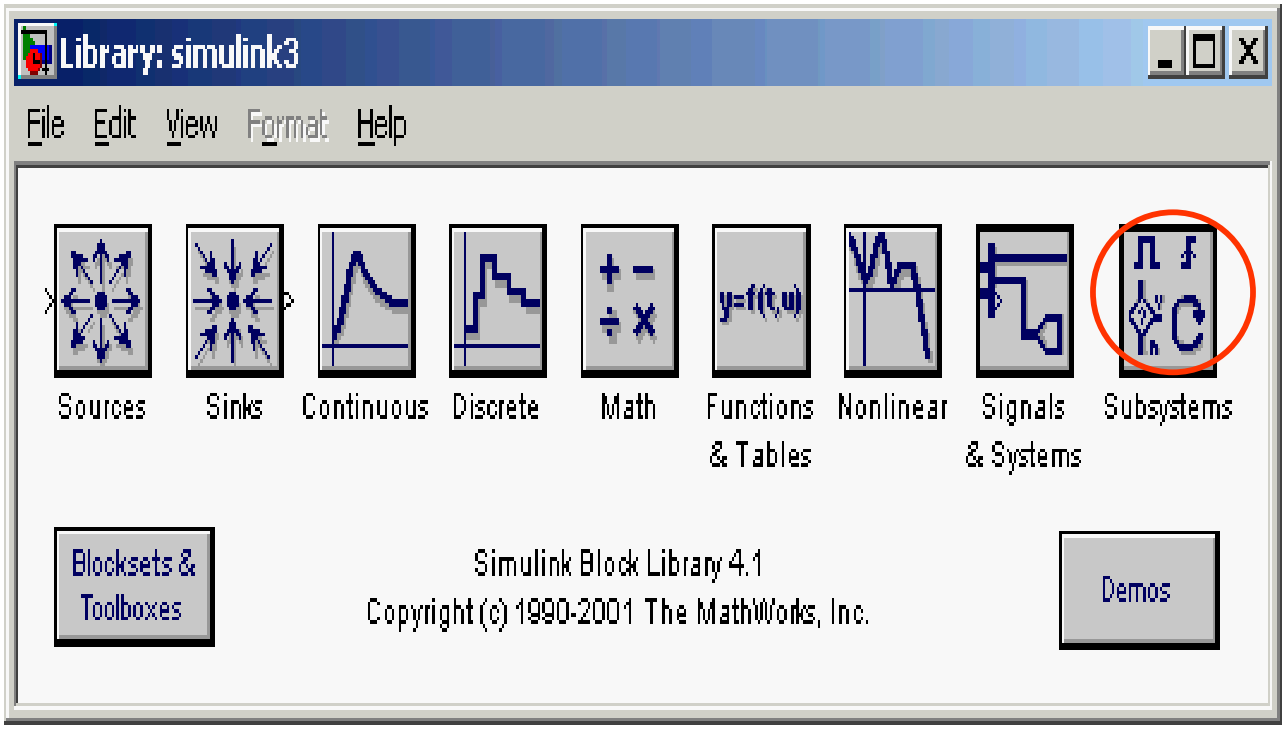
Icône SIMULINK



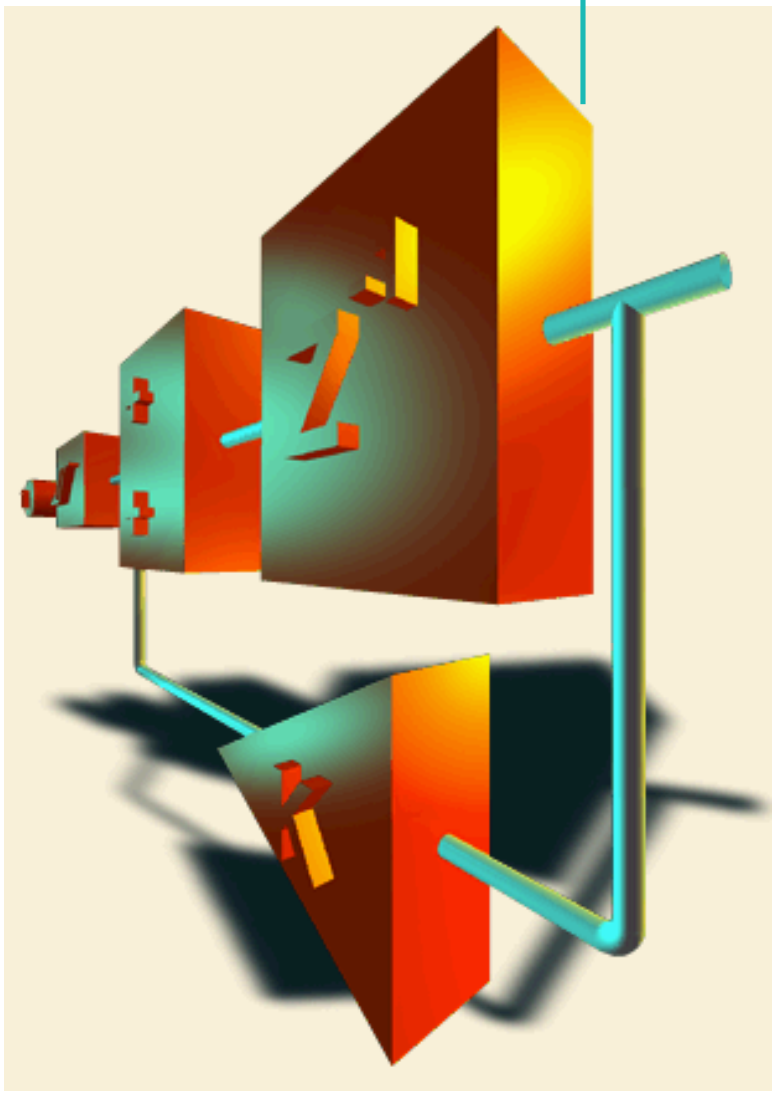
1.6 Exécution de Simulink (cont.)

Pour avoir Simulink sous forme icônographique taper :

```
>> simulink3
```

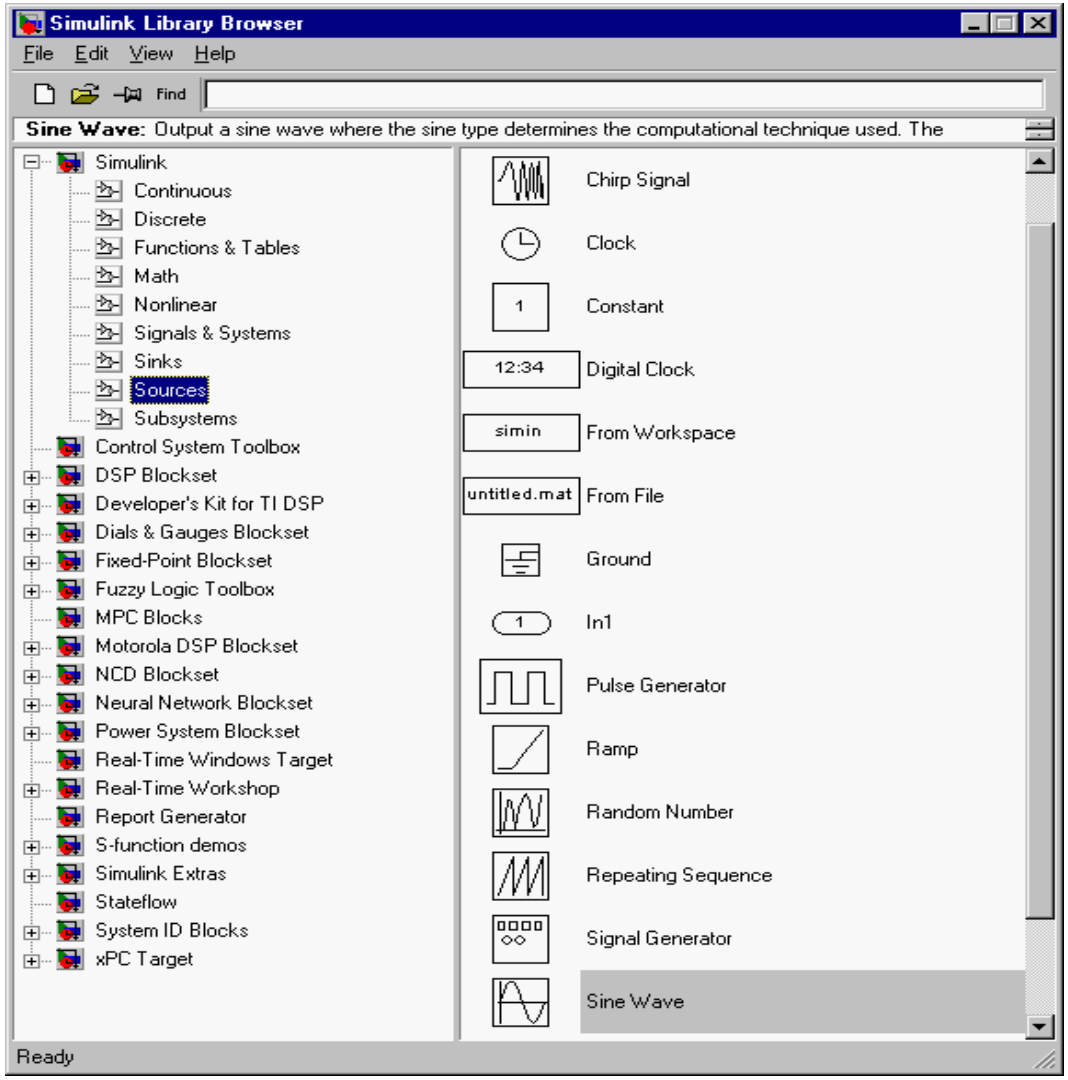


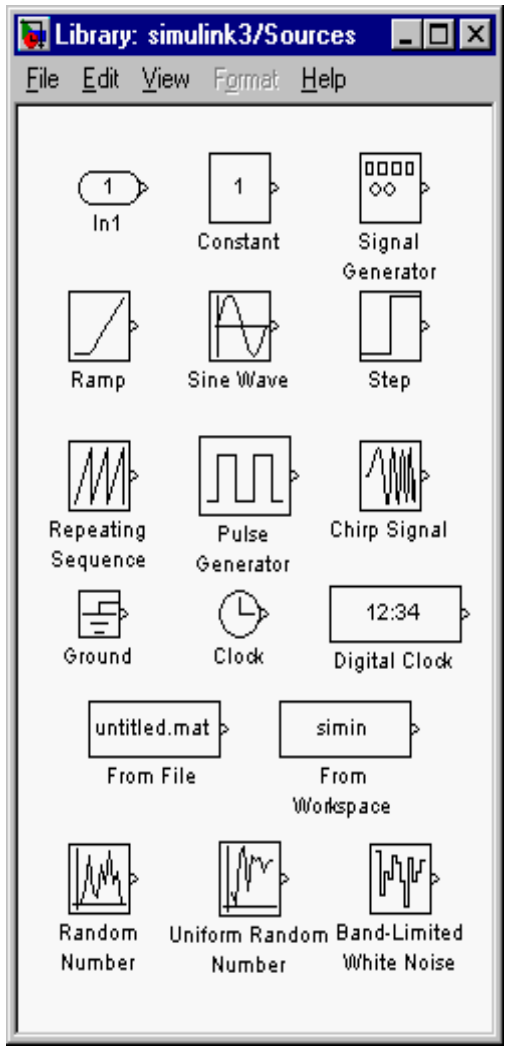
2. Les Bibliothèques de Simulink



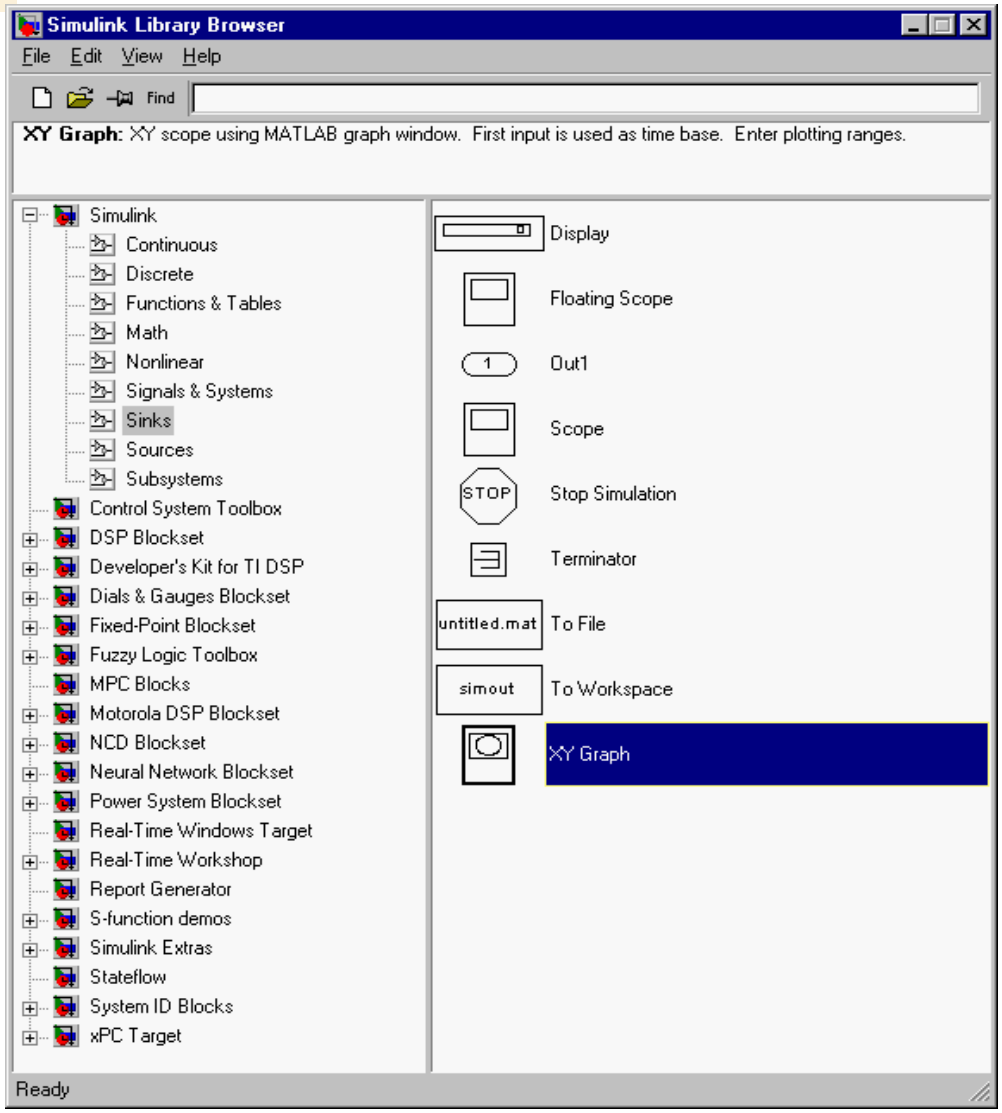
2.1 Génération des Signaux

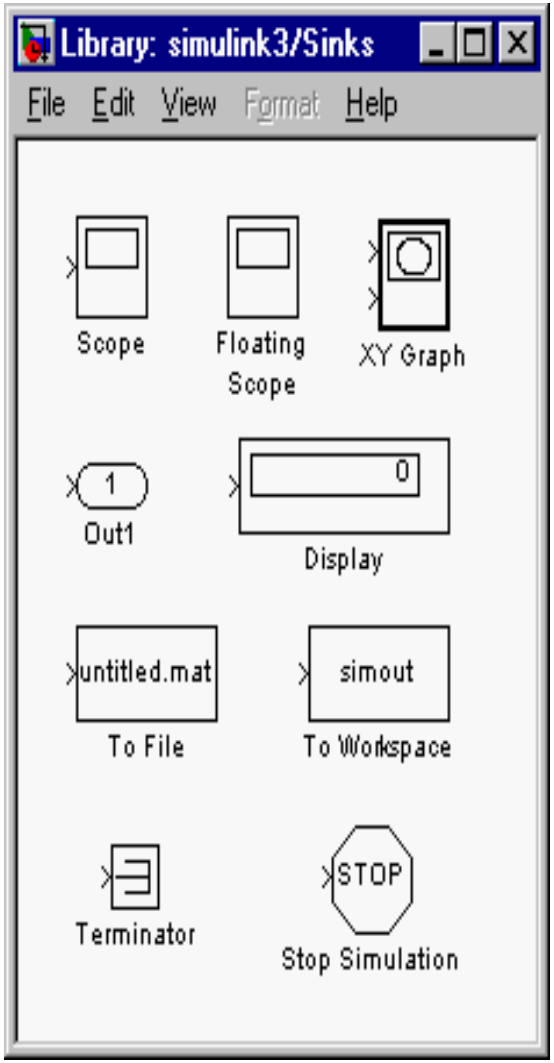
- **Pour ouvrir une bibliothèque, double cliquer sur les blocs :**



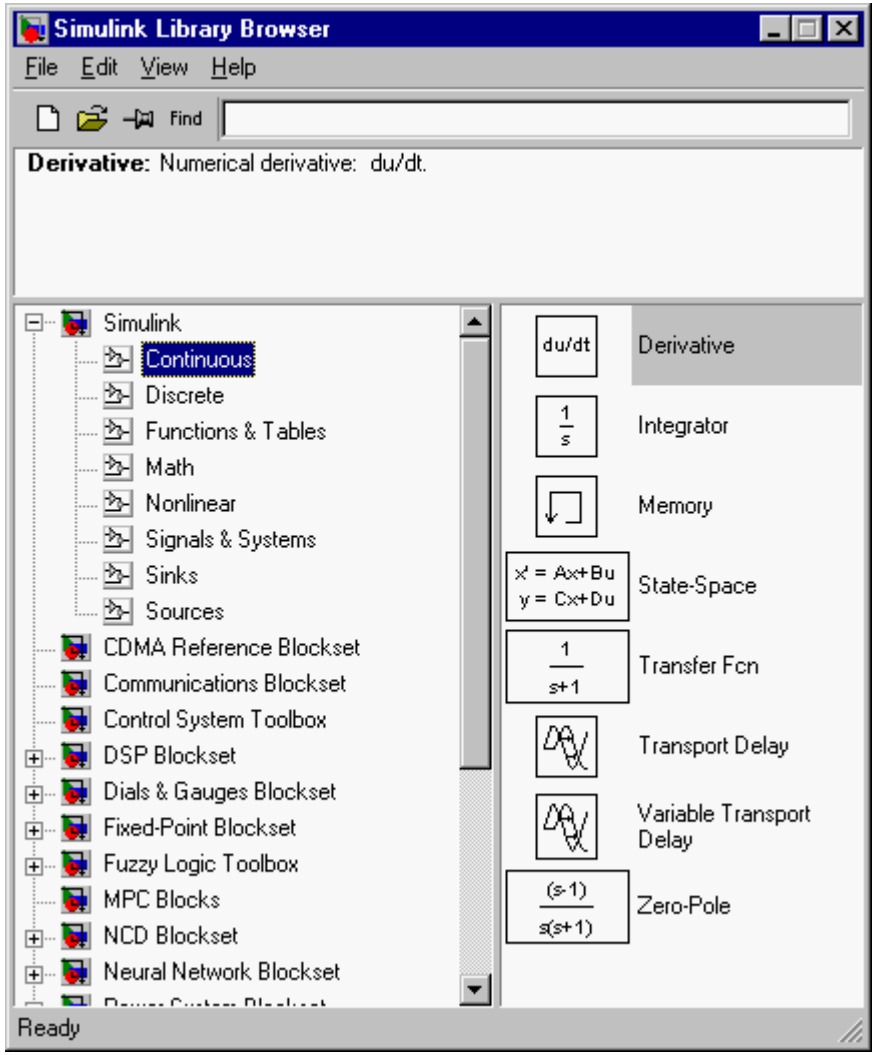


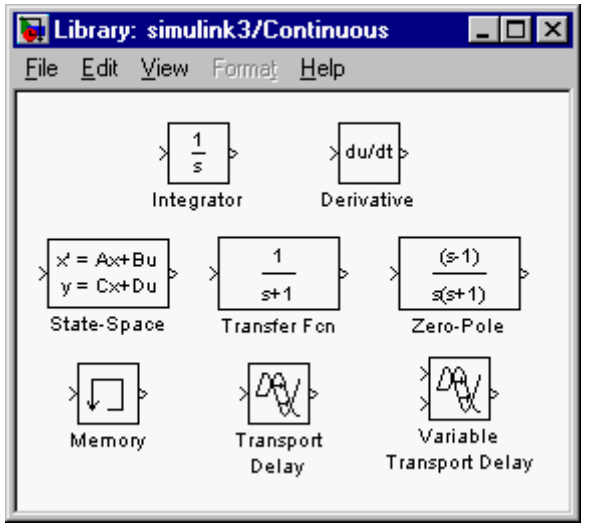
2.2 Visualisation des Signaux



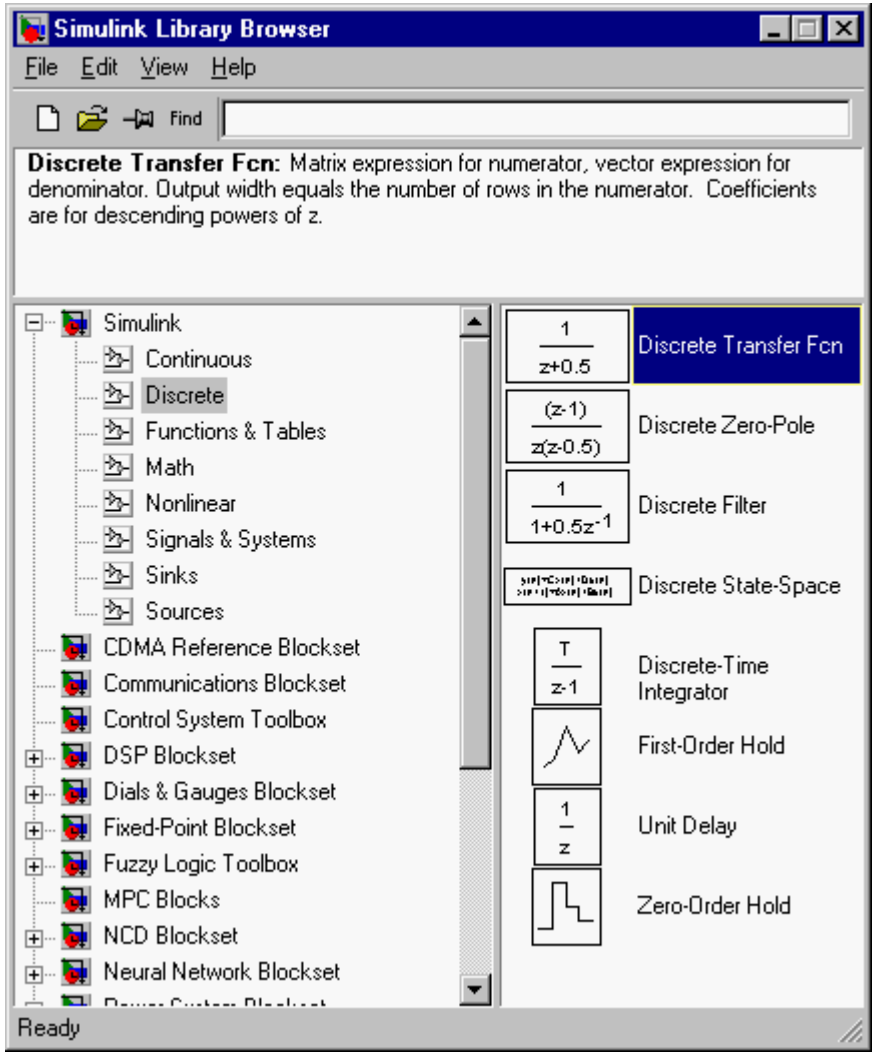


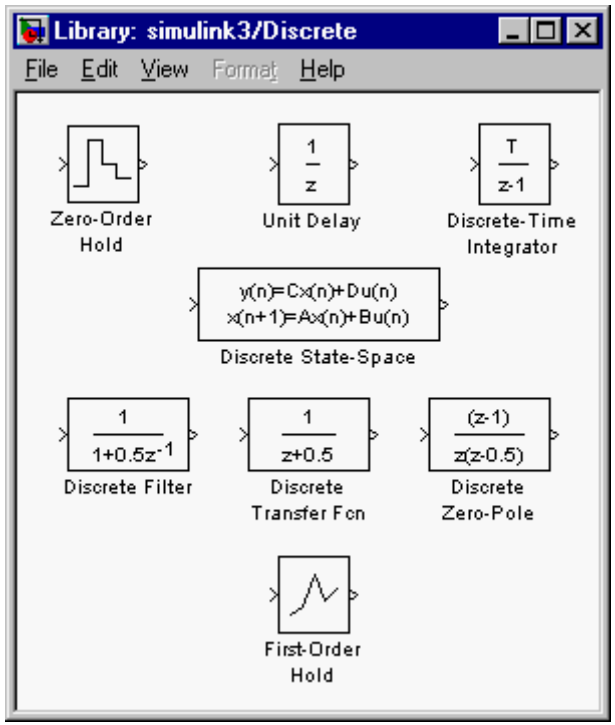
2.3 Systèmes Continus



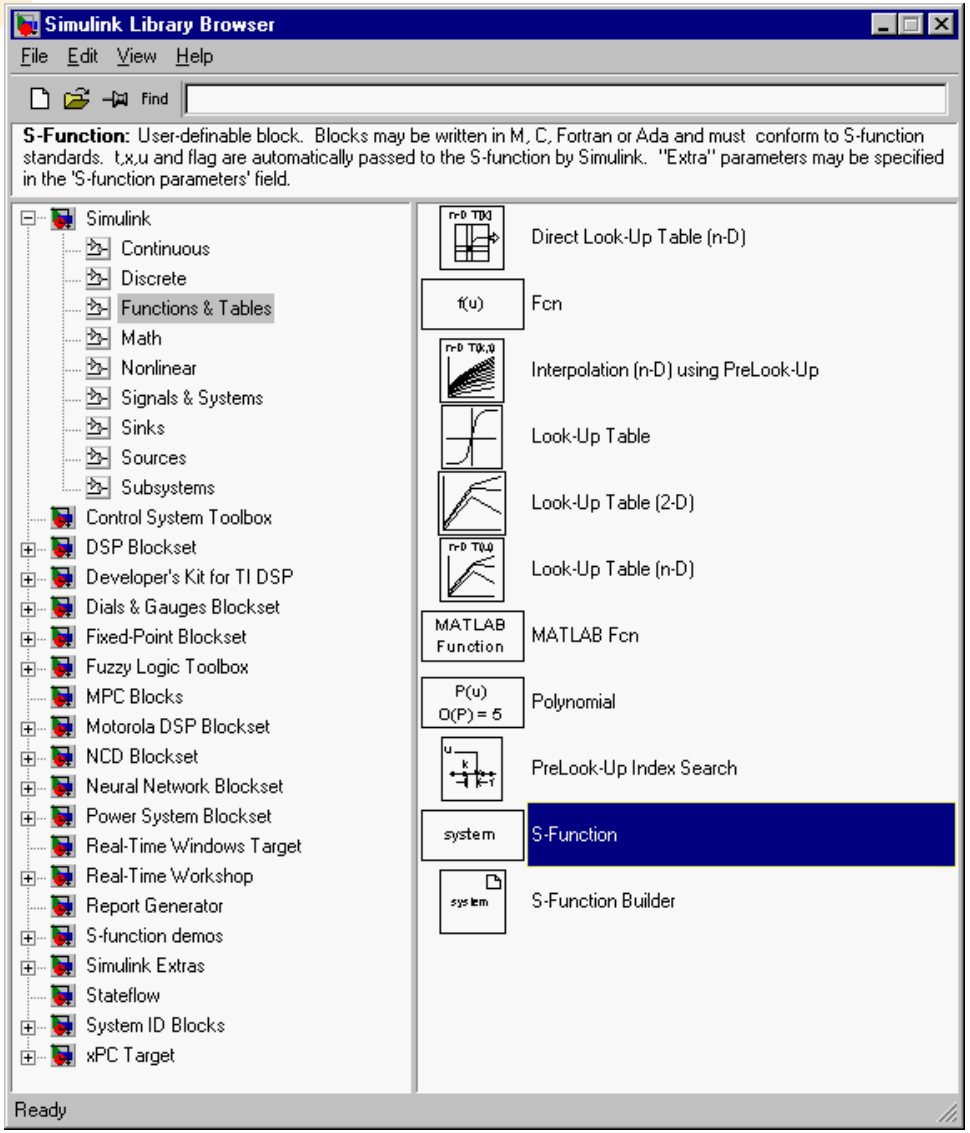


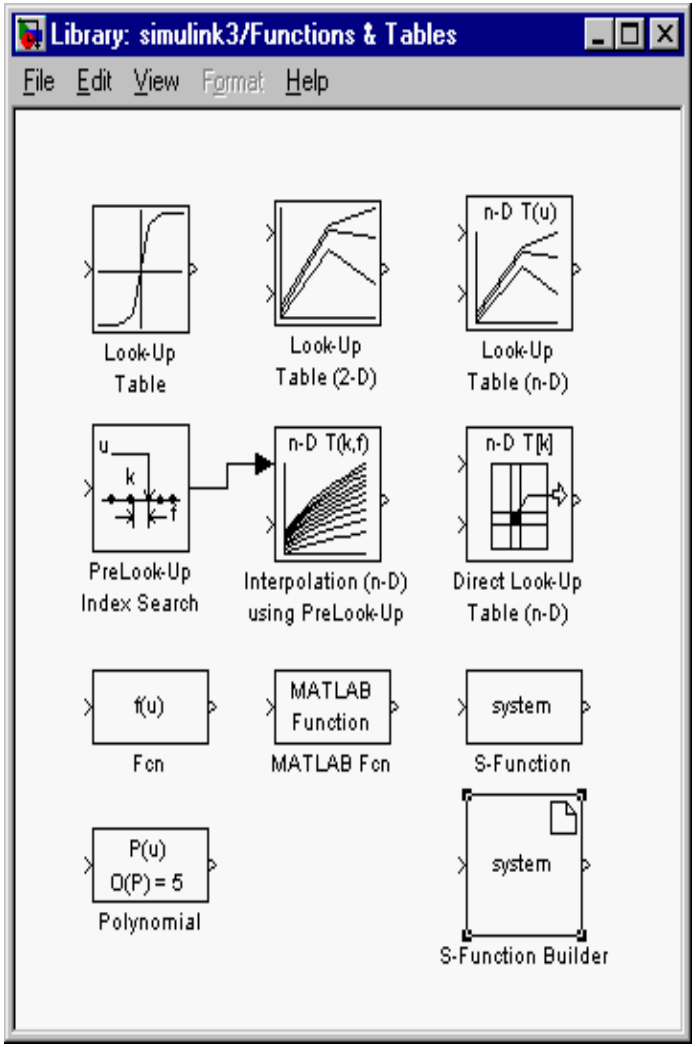
2.4 Systèmes Discrets



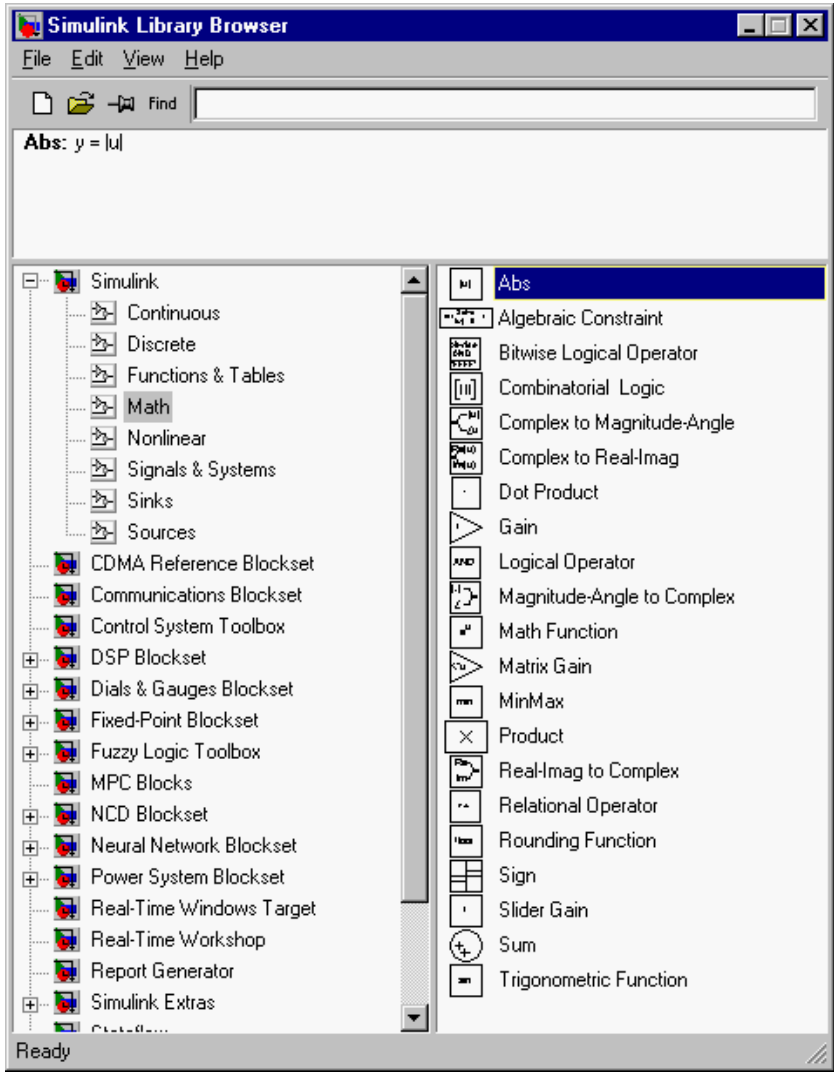


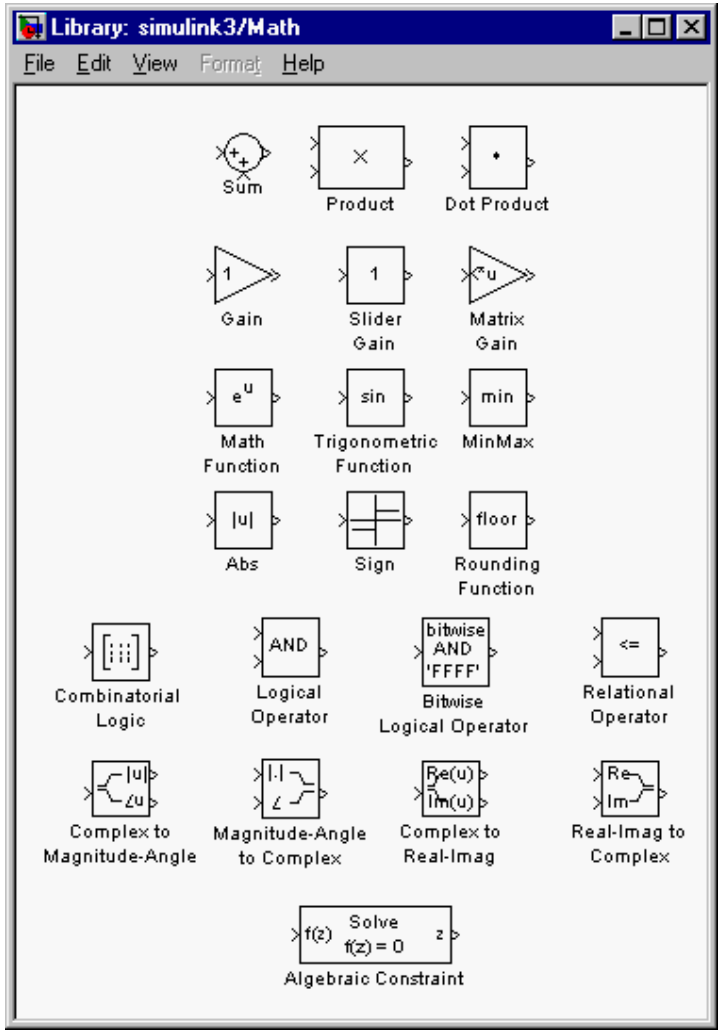
2.5 Fonctions et Tables



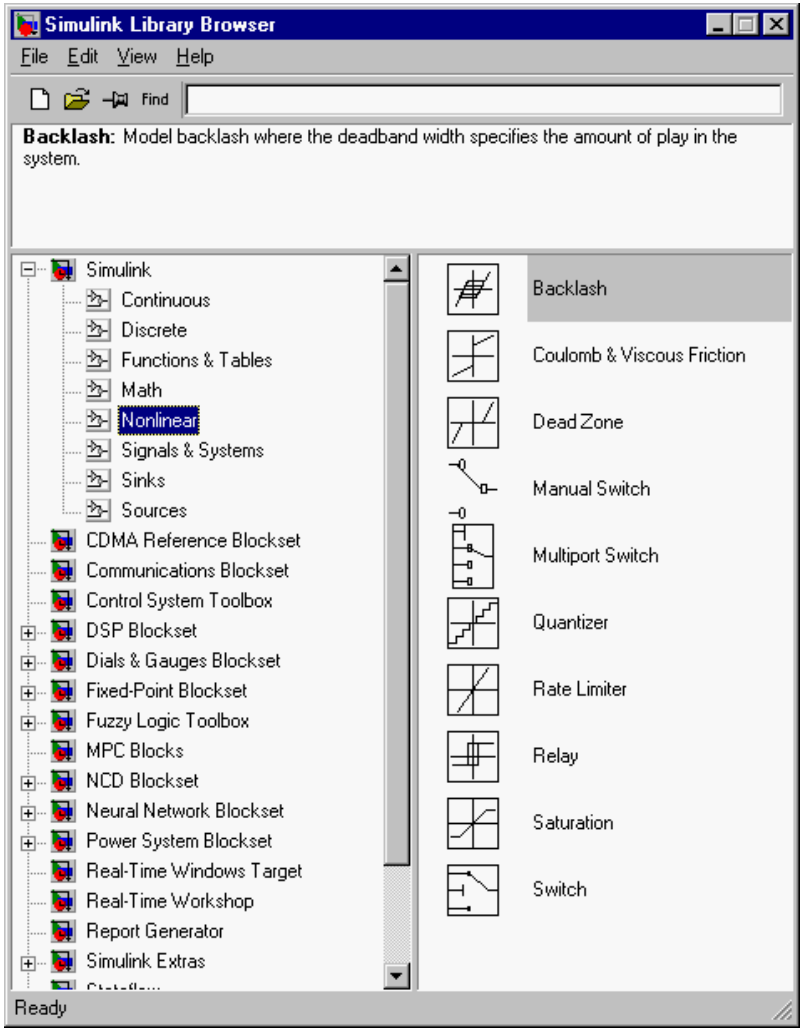


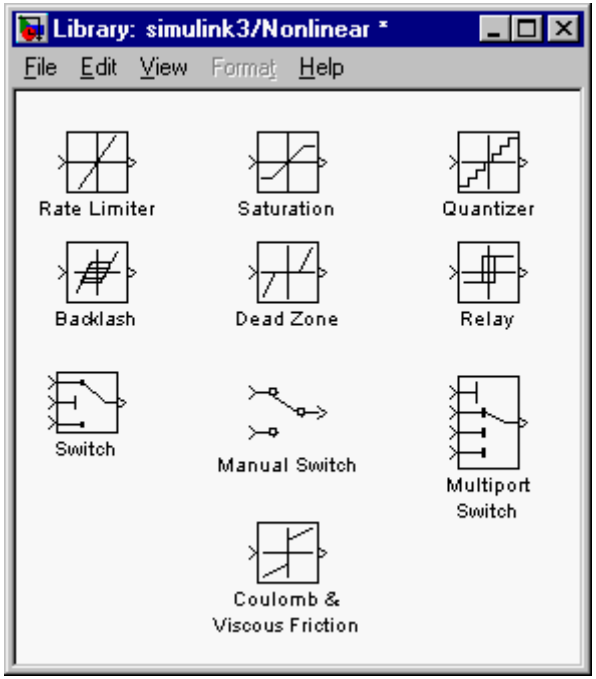
2.6 Opérations Mathématiques



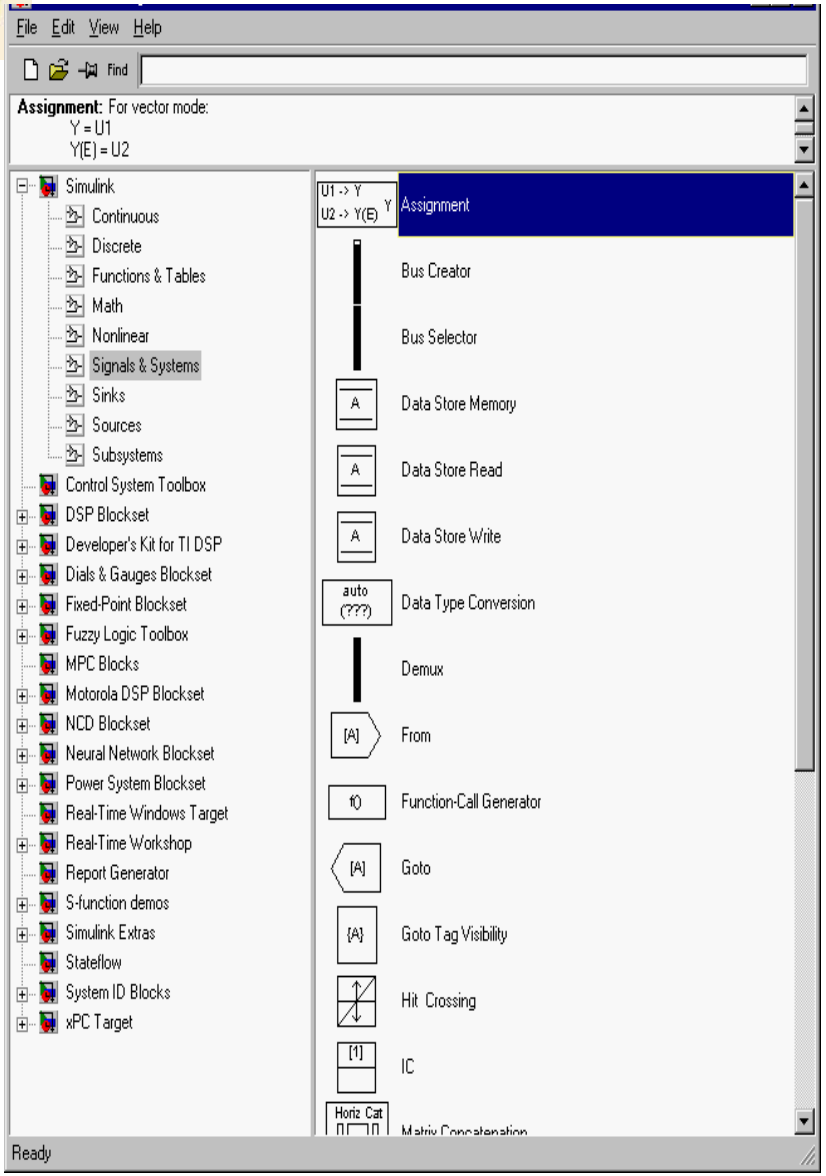


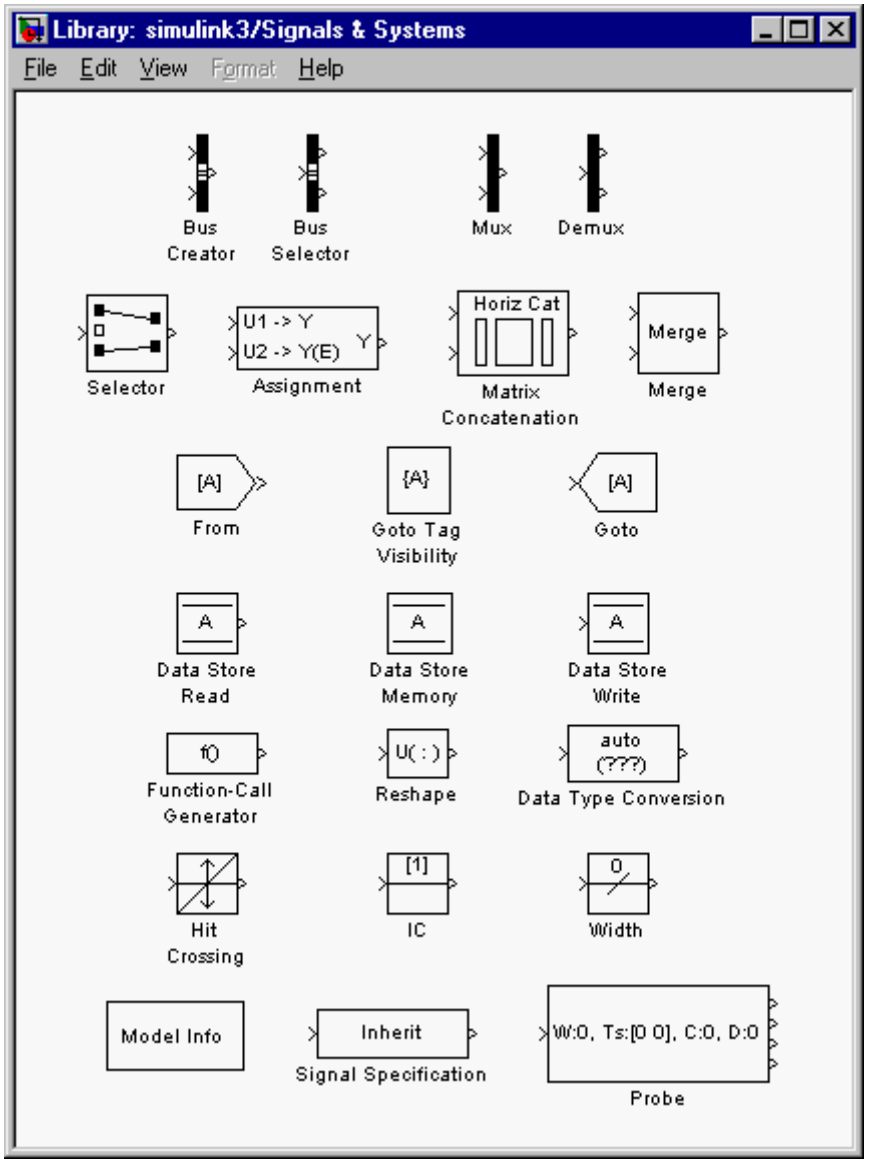
2.7 Non-Linéarités





2.8 Gestion des Signaux & Systèmes



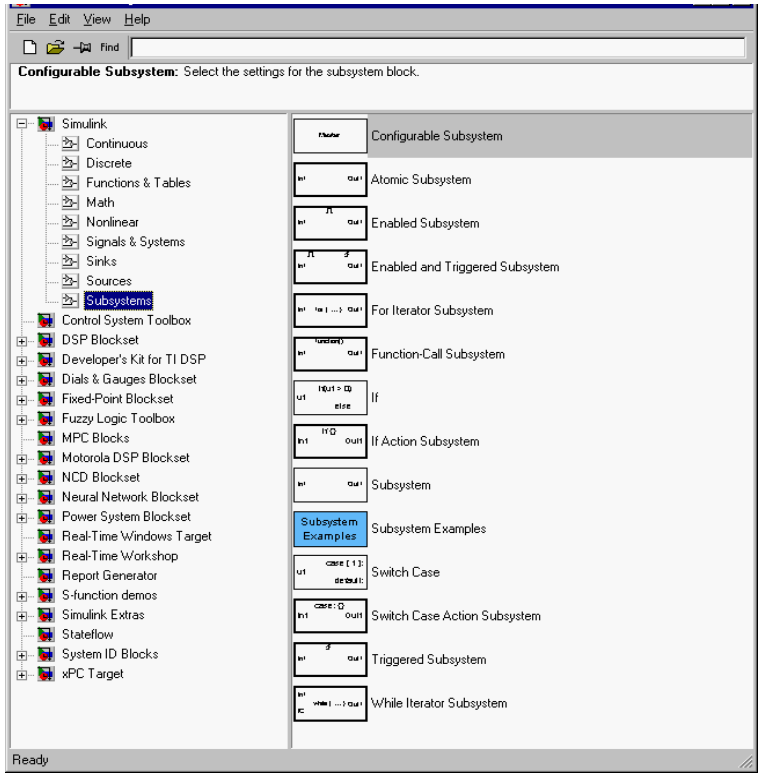


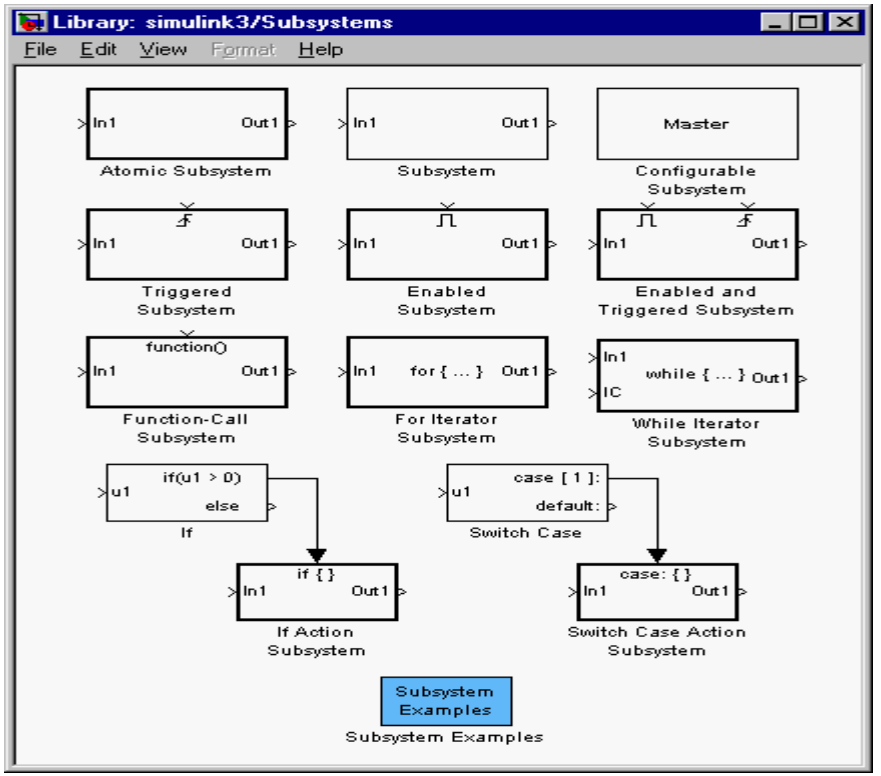
2.9 Classes de Sous-Systèmes

Il existe 2 classes des sous-systèmes

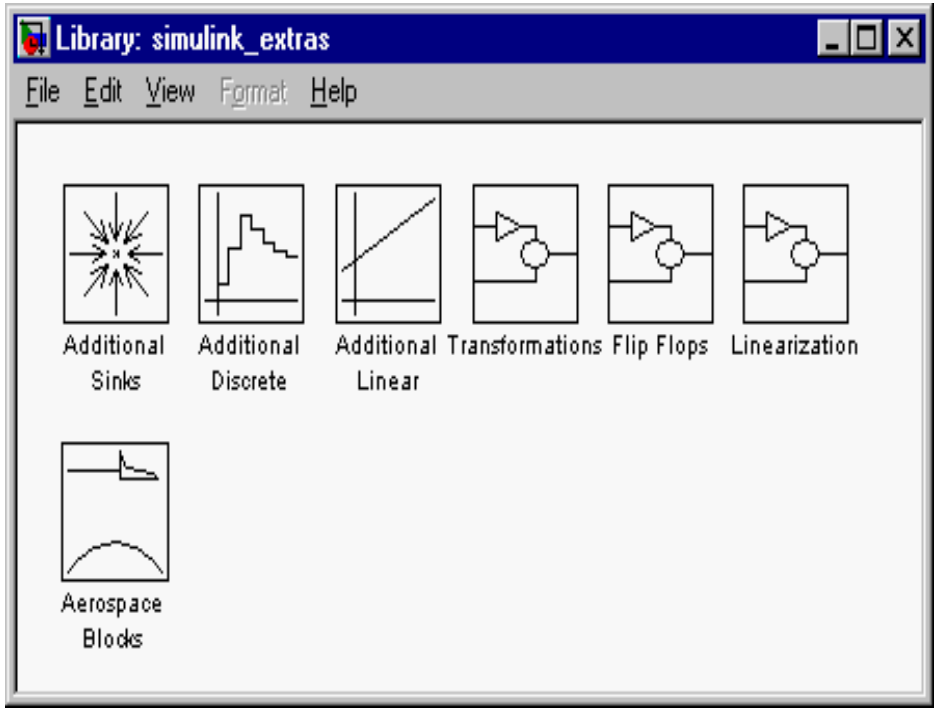
1) sous-systèmes virtuels: assurent une hiérarchie graphique dans les modèles. Ils n'influent pas sur l'exécution du modèle. Lors de l'exécution du modèle, Simulink aplatit tous les sous-systèmes virtuels.

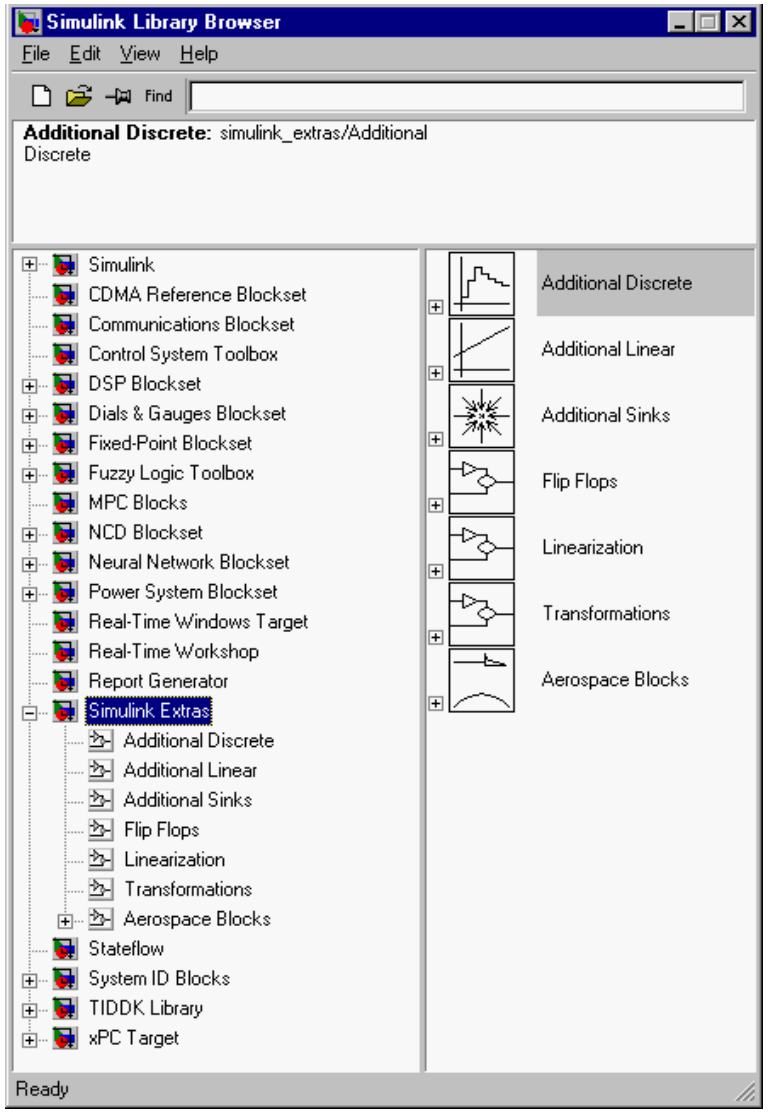
2) sous-systèmes non-virtuels: assurent également la hiérarchie graphique et altèrent le mode d'exécution du modèle . Le sous-système non-virtuel est exécuté comme une unité simple (exécution atomique). Tous les sous-systèmes non-virtuels sont dessinés avec un cadre gras.





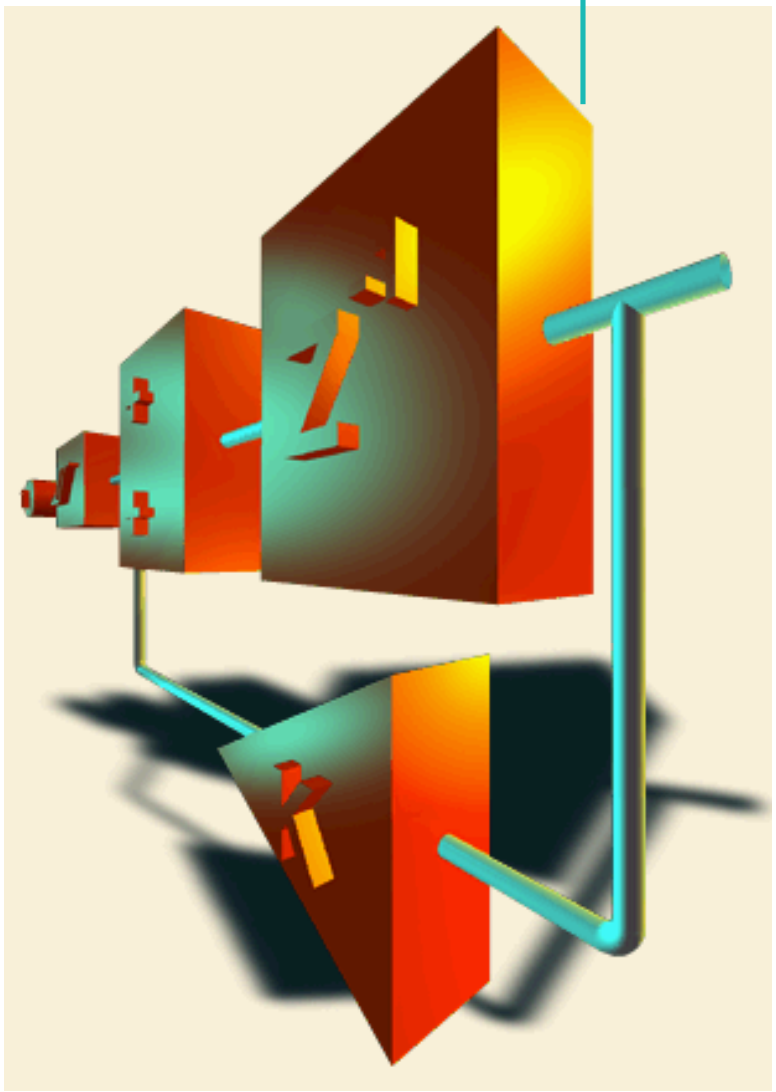
2.10 Blocs Additionnels





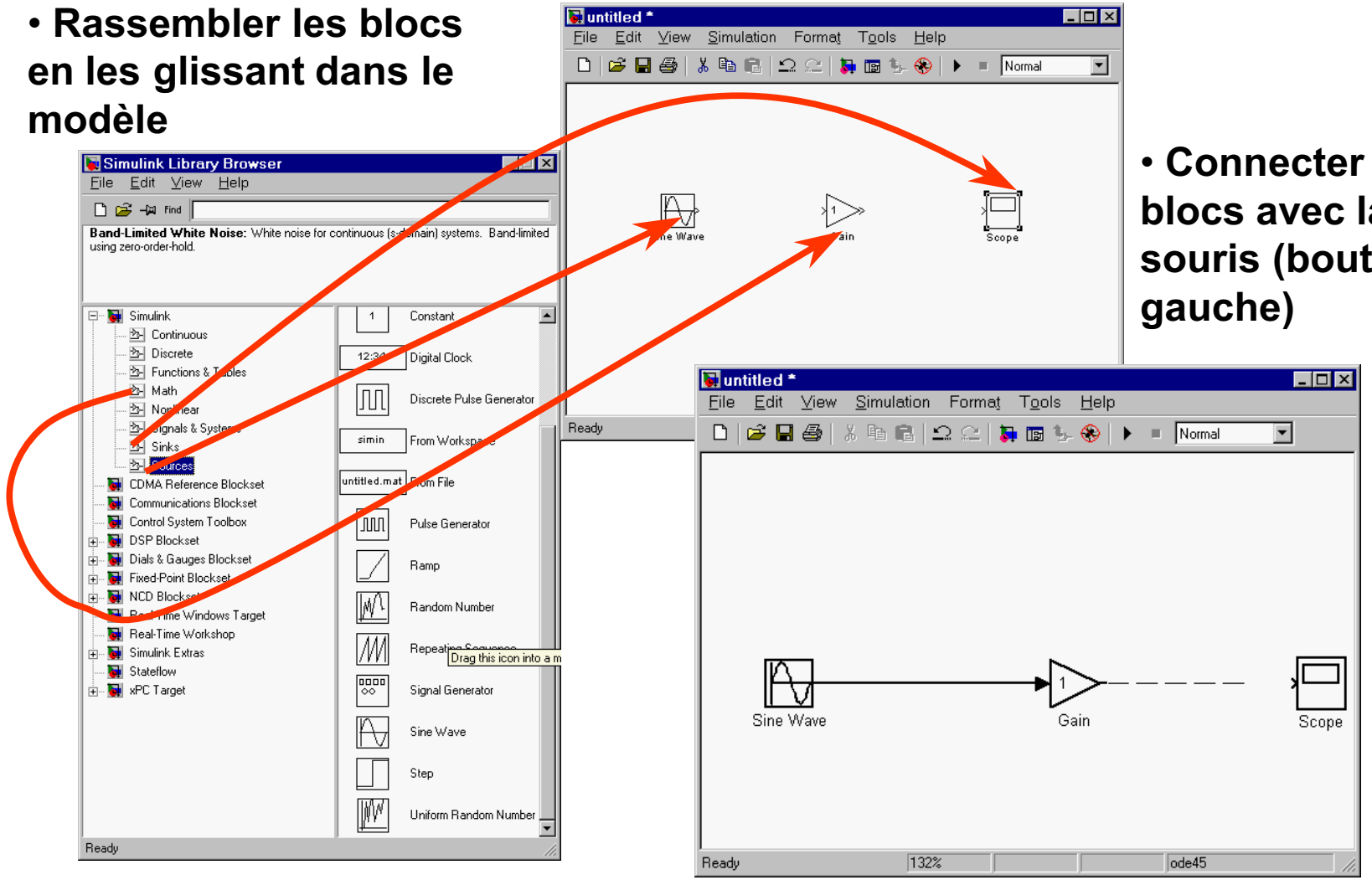
Simulink

3. Construction de Systèmes



3.1 Collection et Connection des Blocs

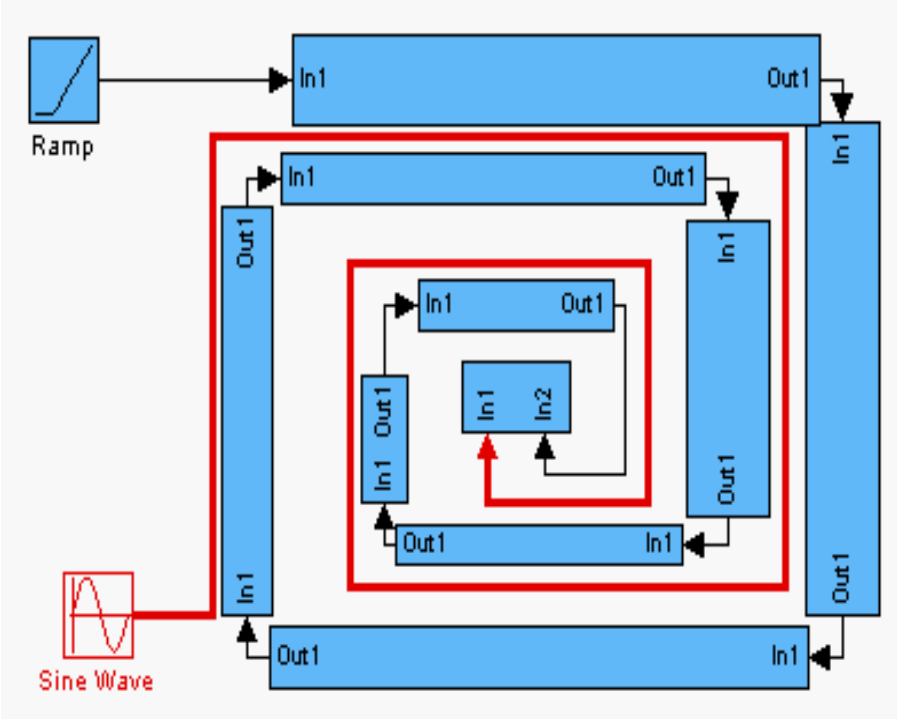
- **Rassembler les blocs en les glissant dans le modèle**



- **Connecter les blocs avec la souris (bouton gauche)**

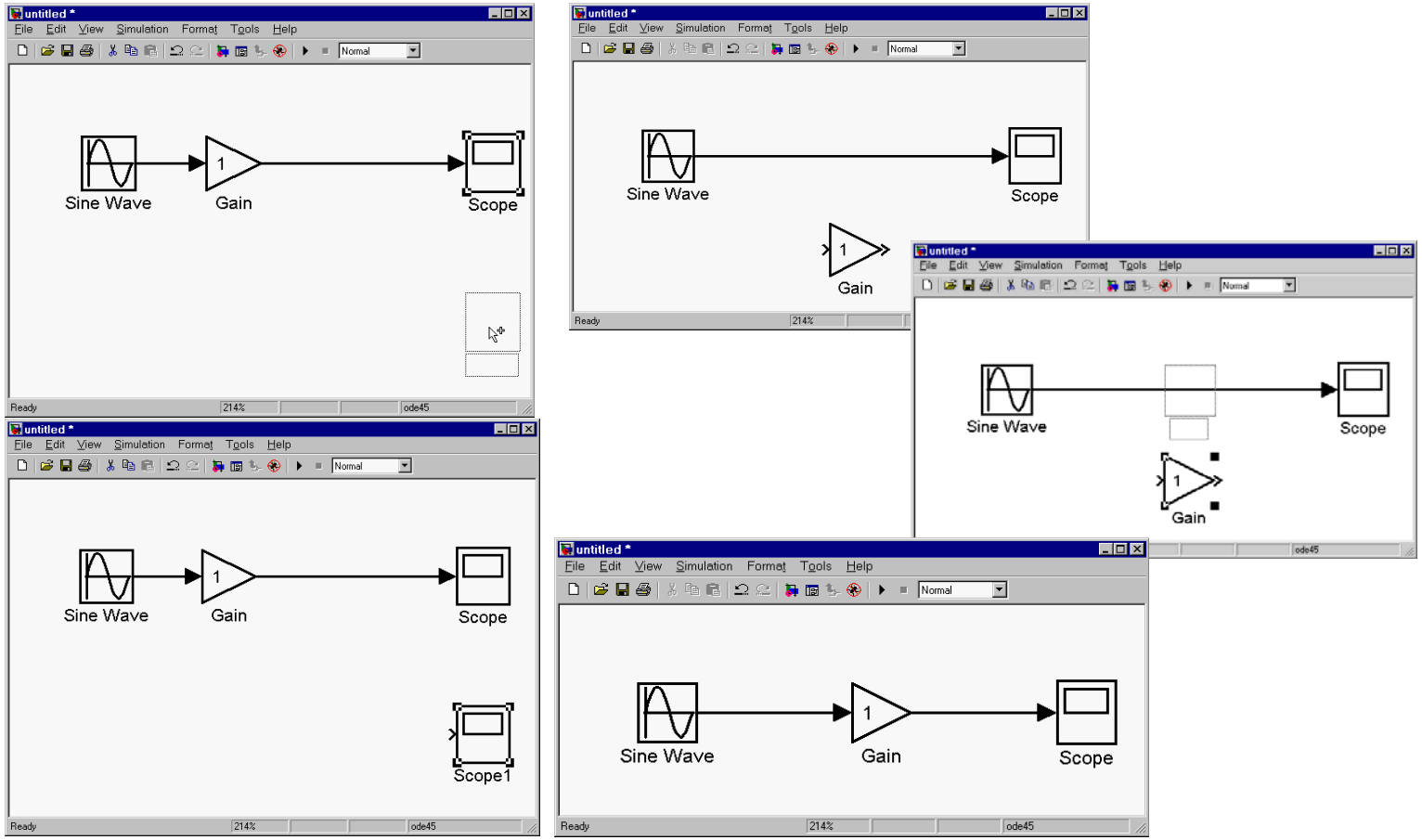
3.1 Collection et Connection des Blocs

• **Pour relier un ensemble de blocs sources (ou cibles) à un bloc cible (ou source), il suffit de sélectionner les blocs sources (ou cibles), de maintenir « Ctrl » enfoncé et effectuer un clic « bouton gauche » sur le bloc cible (ou source).**



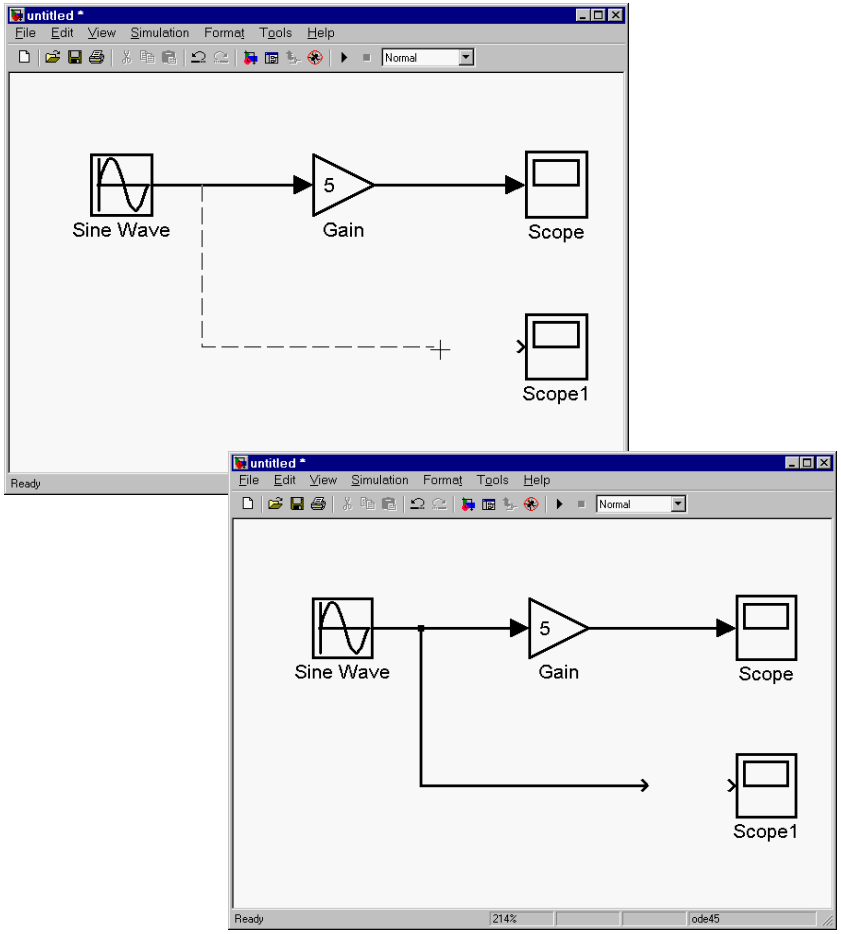
3.2 Copie et Insertion de Blocs

- **Clic bouton droit de la souris sur le bloc pour le copier**
- **Placer le bloc au-dessus de la ligne d'insertion**

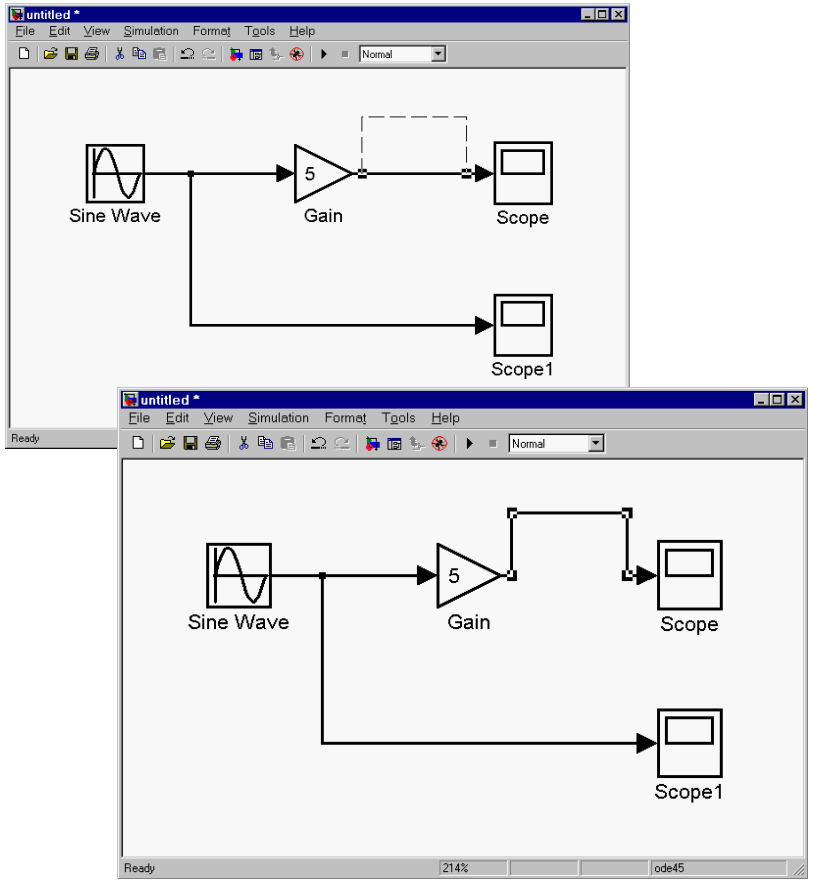


3.3 Branchement et Cheminement des Lignes

- **Clic bouton droit de la souris sur la ligne pour ajouter une branche**

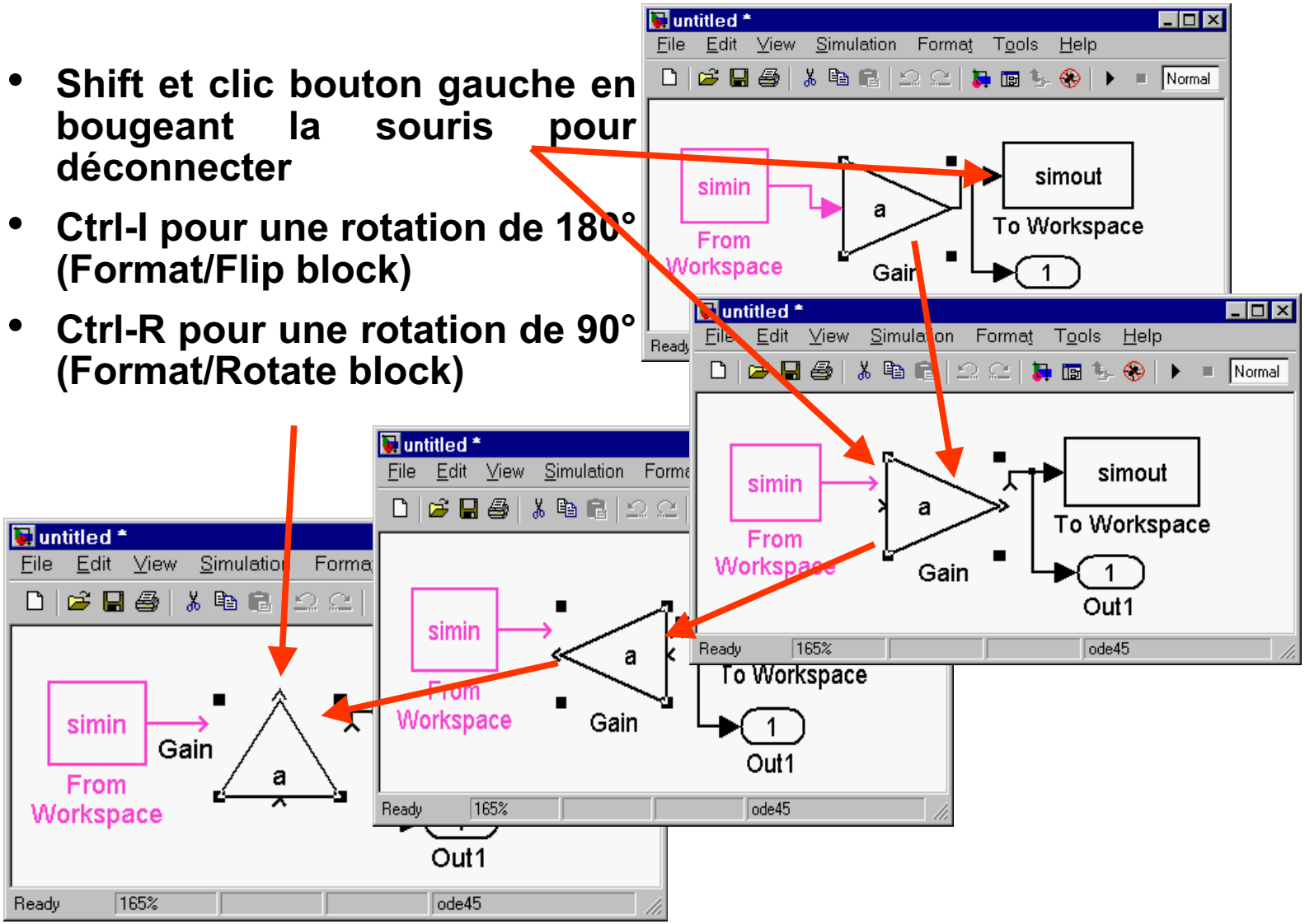


- **Clic bouton gauche de la souris sur la ligne pour recheminement**



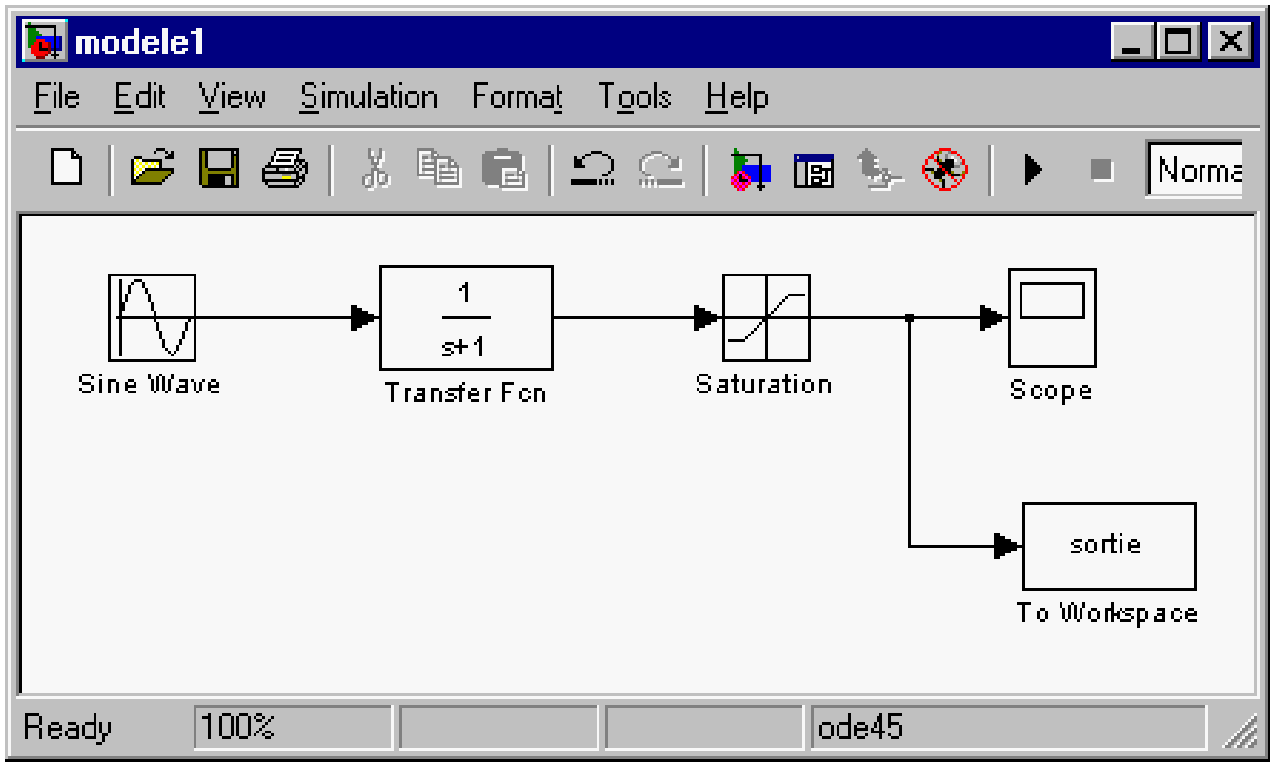
3.4 Changement de Position/Orientation

- **Shift et clic bouton gauche en bougeant la souris pour déconnecter**
- **Ctrl-l pour une rotation de 180° (Format/Flip block)**
- **Ctrl-R pour une rotation de 90° (Format/Rotate block)**



3.5 Mise en Place des Blocs

- **Voici le modèle à construire :**



3.5 Mise en Place des Blocs (cont.)

- **Créer un nouveau modèle : File/New/Model de la fenêtre Simulink**

- **Ajouter un bloc : cliquez sur le bloc désiré dans la bibliothèque et faites le glisser vers la fenêtre où vous voulez placer ce bloc.**

- **Pour relier deux blocs, cliquez sur la sortie du premier bloc et faites glisser la souris jusqu'à l'entrée du second bloc, puis relâchez la souris. Cette manipulation peut aussi se faire en partant de l'entrée du second bloc et en allant vers la sortie du premier. En appuyant sur la touche SHIFT pendant la manipulation , l'extrémité du fil suit exactement la souris.**

- **Pour redimensionner un bloc, cliquez dans un coin du bloc et déplacez la souris.**

3.5 Mise en Place des Blocs (cont.)

- **Pour redimensionner un bloc, cliquez dans un coin du bloc et déplacez la souris.**

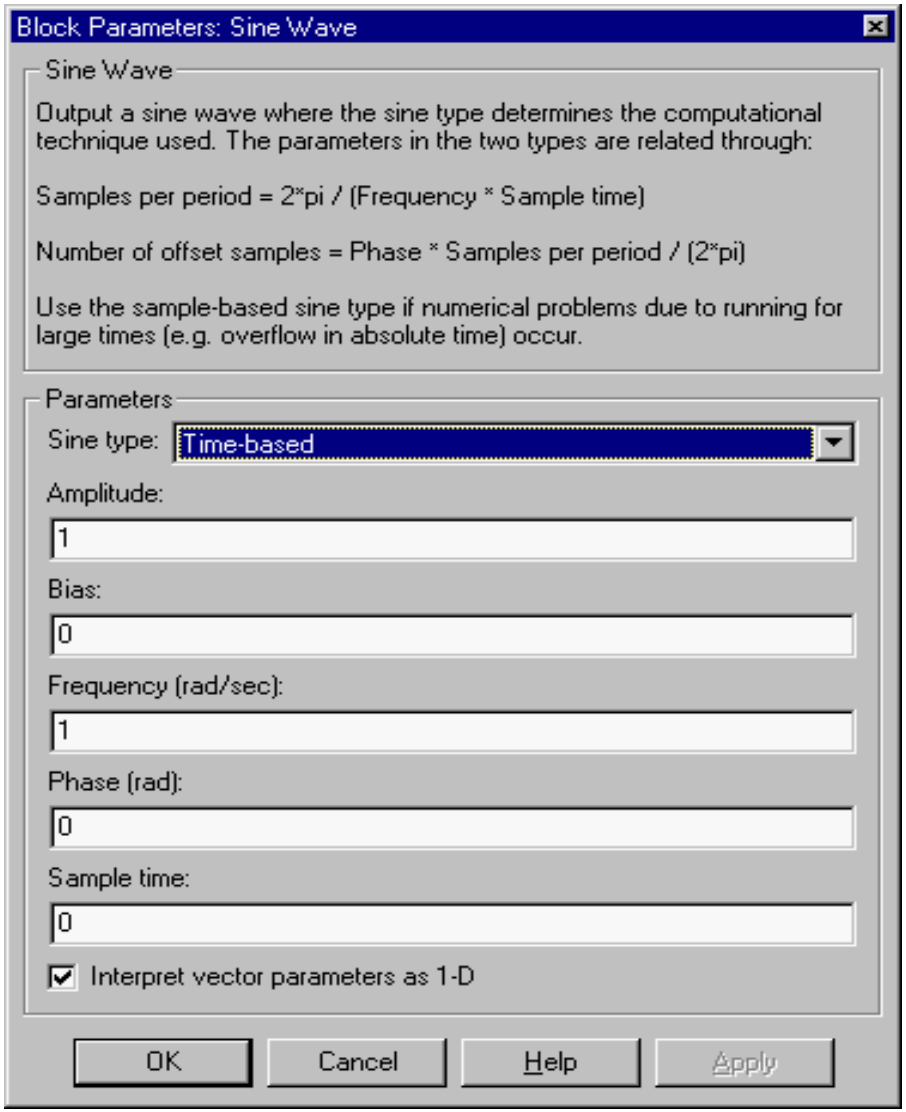
- **Pour copier un bloc, cliquez avec le bouton droit de la souris sur le bloc à copier. Faites glisser la souris. Le bloc est copié à l'endroit où le bouton de la souris est relâché.**

- **Pour paramétrer un bloc, double-cliquez (bouton de gauche) sur le bloc concerné ou cliquez une fois sur le bouton de droite. Un menu apparaît où le champ « Parameters » est alors accessible. Par cette deuxième méthode, vous pouvez également sélectionner la rubrique « Properties » pour personnaliser le bloc.**

3.6 Paramétrage des Blocs

Le bloc *Sine Wave*

Le sinus est d'amplitude 1, de fréquence 1 rad/s avec une phase nulle. La période d'échantillonnage de 0 indique que nous sommes dans le domaine continu.



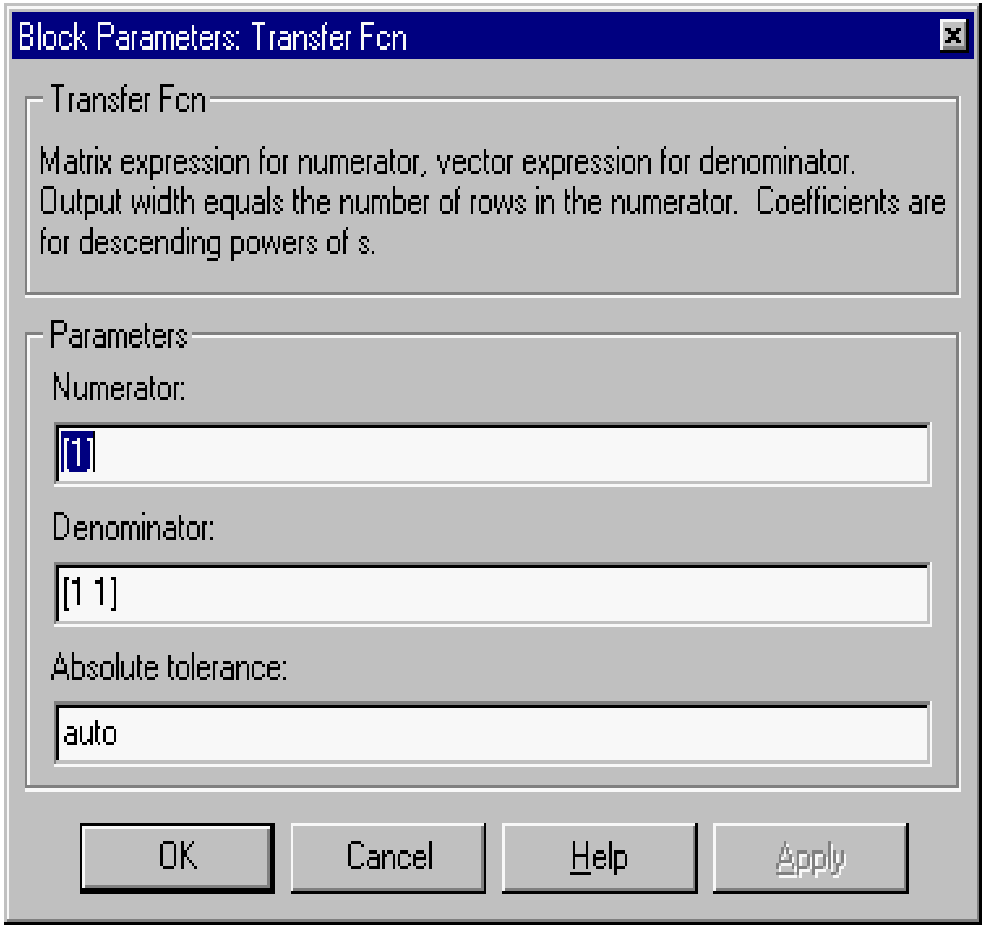
3.6 Paramétrage des Blocs (cont.)

Le bloc *Transfer Fcn*

La fonction de transfert est :

$$\frac{1}{s+1}$$

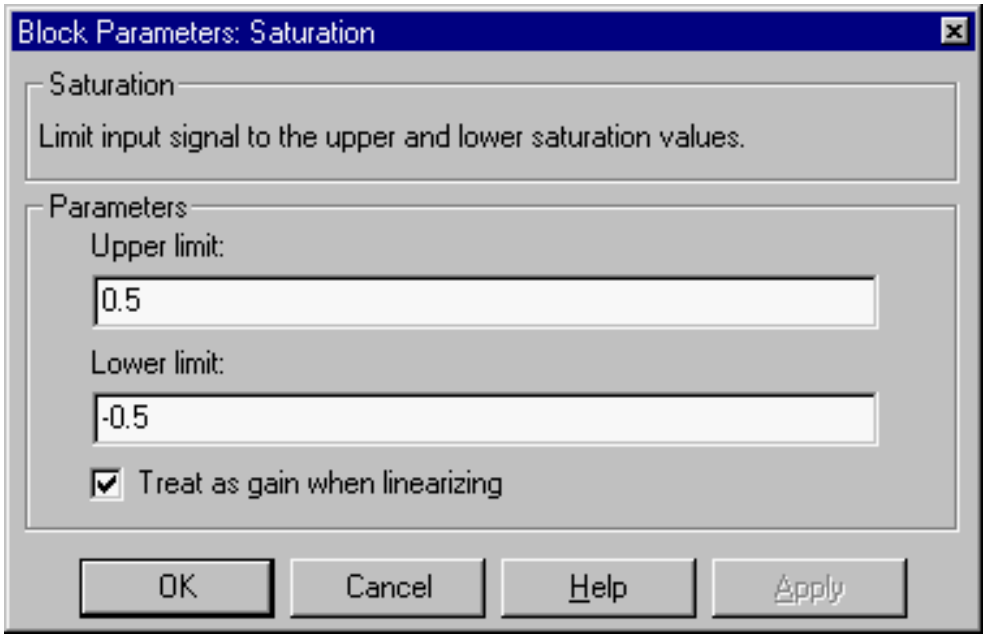
Les coefficients du numérateur et du dénominateur sont entrés sous forme de vecteurs par puissance décroissante comme dans MATLAB.



3.6 Paramétrage des Blocs (cont.)

Le bloc *Saturation*

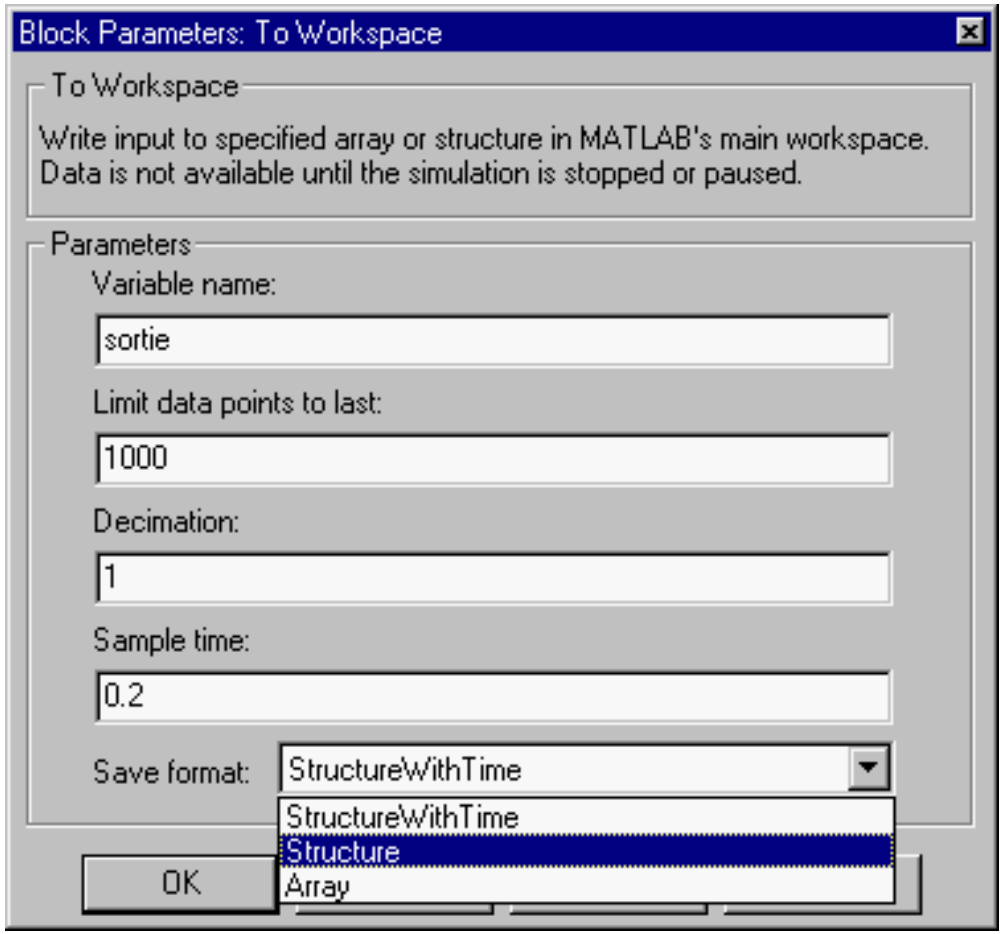
Il limite la sortie entre les valeurs -0.5 et 0.5.



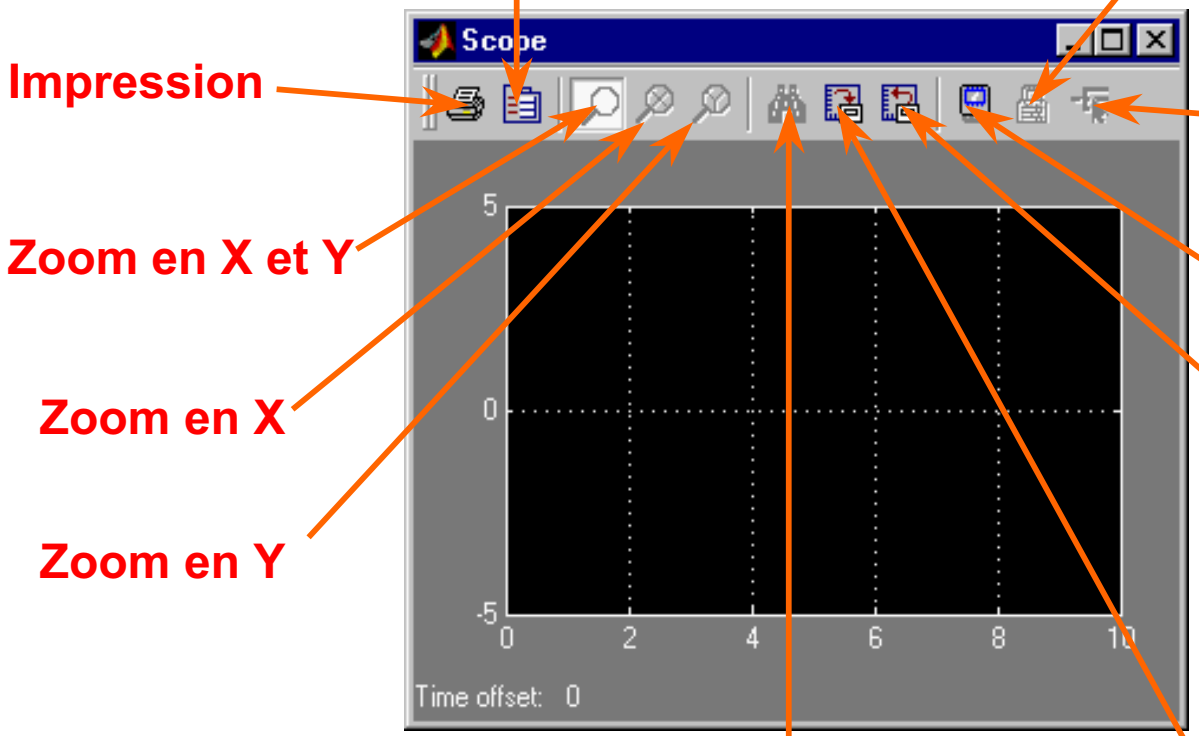
3.6 Paramétrage des Blocs (cont.)

Le bloc *To Workspace*

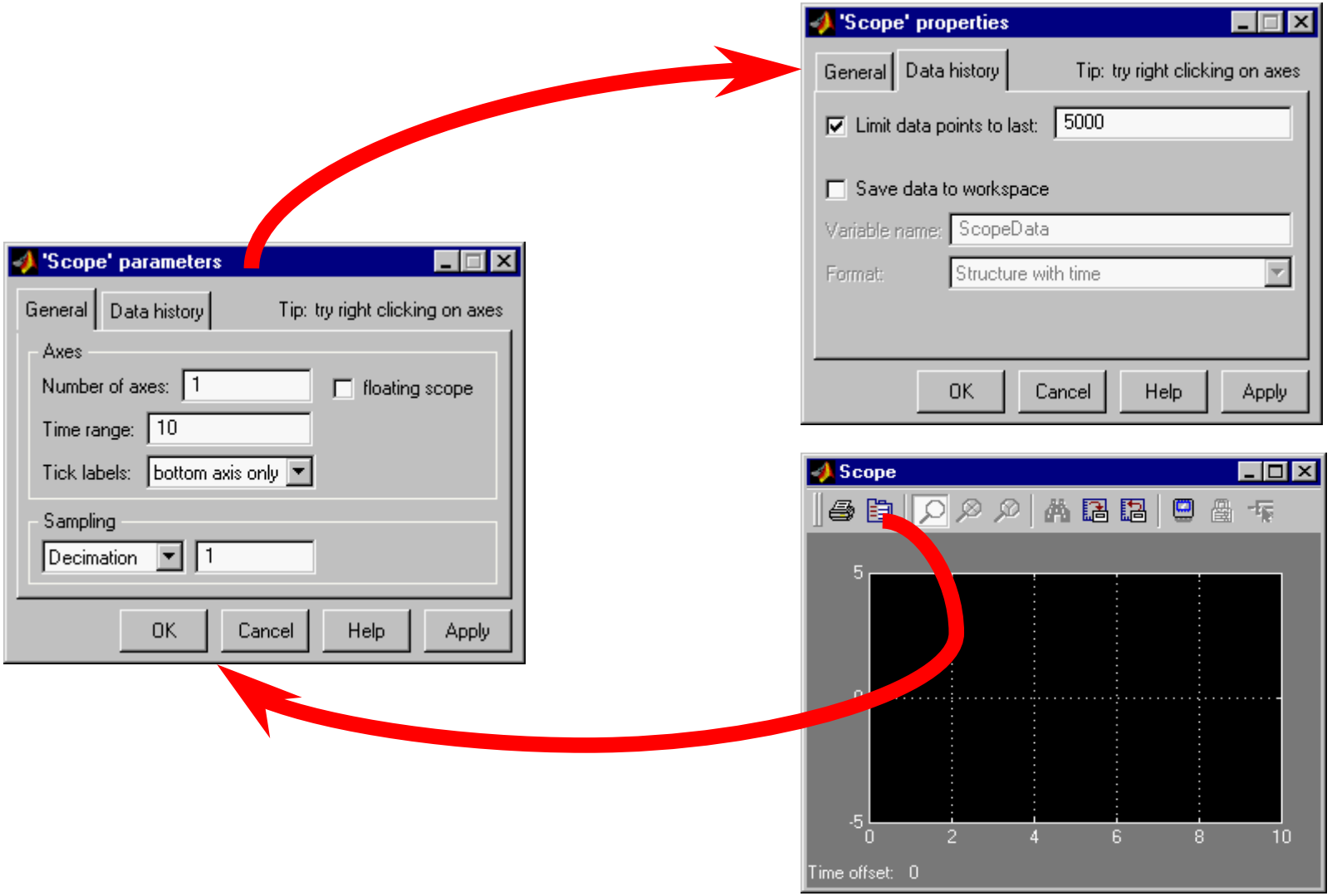
Les 1000 derniers points de la simulation sont stockés dans la variable sortie dont le format est au choix une structure ou une matrice.



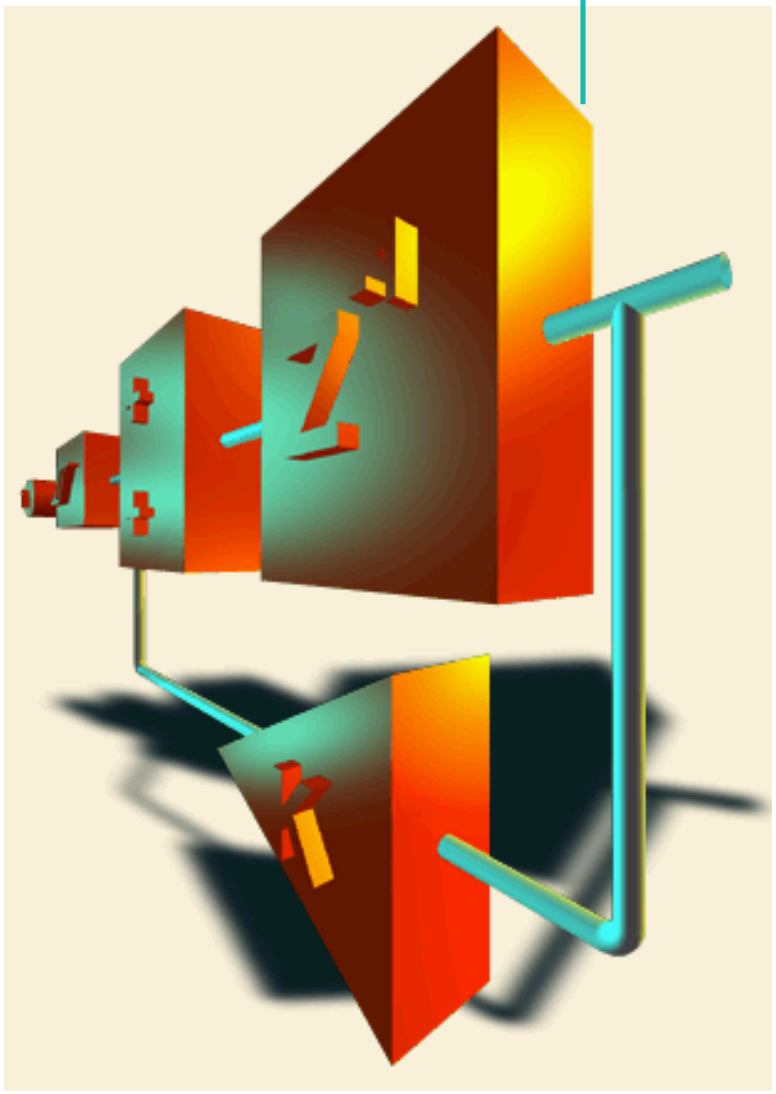
3.6 Paramétrage des Blocs (cont.)



3.6 Paramétrage des Blocs (cont.)



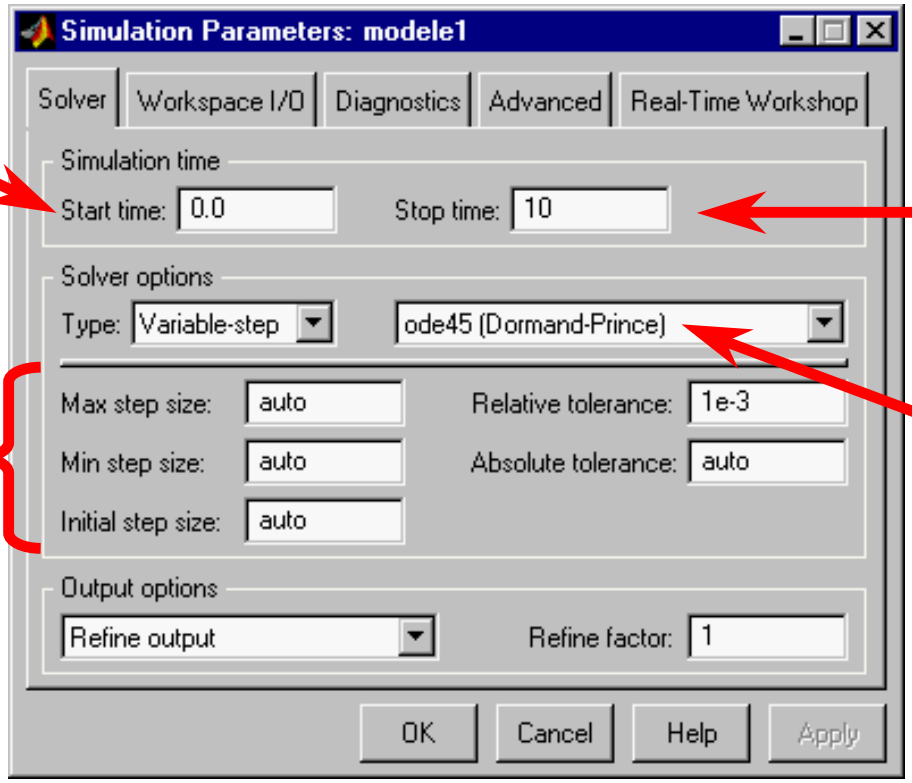
4. Simulation



4.1 Paramètres de Simulation

Les paramètres de la simulation sont saisis dans la fenêtre ci-dessous obtenue par le menu « Simulation/Parameters » du système étudié :

Instant de départ de la simulation



Instant final de simulation

Paramètres spécifiques au solver choisi

Choix du Solver (continus, discrets, pas fixe ou variable)

4.1 Paramètres de Simulation (cont.)

Solvers à pas fixe

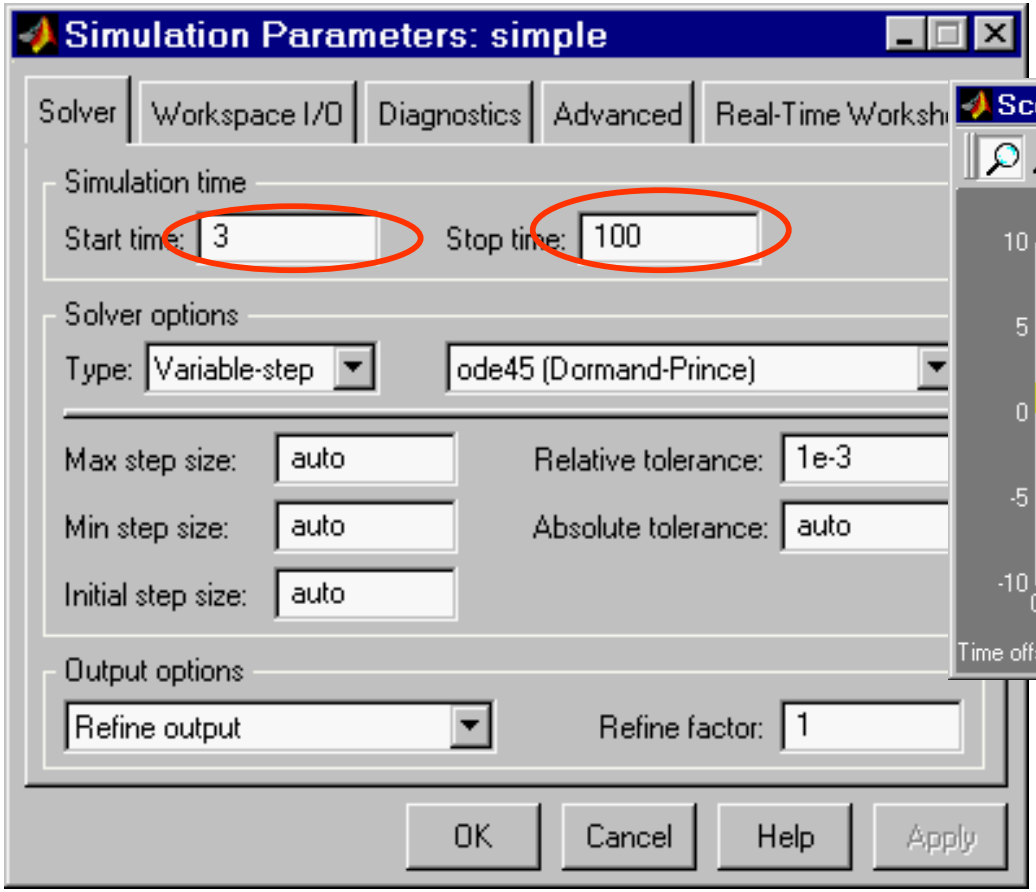
- **Noms :**
  - ***discrete*** (pour les systèmes purement discrets)
  - ***ode5, ode4, ode3, ode2, ode1*** (pour les systèmes hybrides)
- **Paramètres :**
  - ***Fixed step size*** (pas fixe d ’intégration pour toute la simulation)
  - ***Mode*** (mono ou multi-tâches)

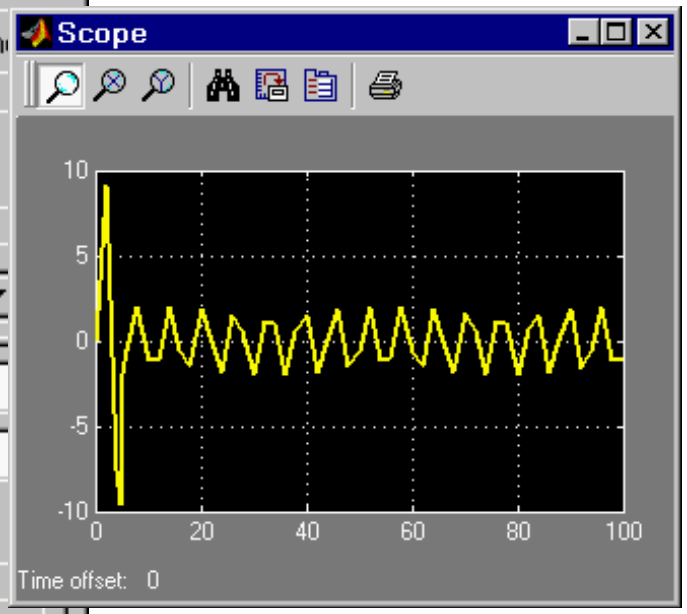
Solvers à pas variable

- **Noms :**
  - ***discrete*** (pour les systèmes purement discrets)
  - ***ode45, ode23, ode113*** (pour les systèmes hybrides non raides)
  - ***ode15s, ode23s, ode23t, ode23tb*** (pour les systèmes hybrides raides)
- **Paramètres :**
  - ***Max step size*** (pas maximum d’intégration)
  - ***Initial step size*** (pas initial de l ’intégration)
  - ***Relative tolerance***
  - ***Absolute tolerance***

Critères d ’adaptation du pas d ’intégration au cours de la simulation

4.1 Paramètres de Simulation





Courbe non-lisse

4.2 Choix du Step Size et des Tolerances

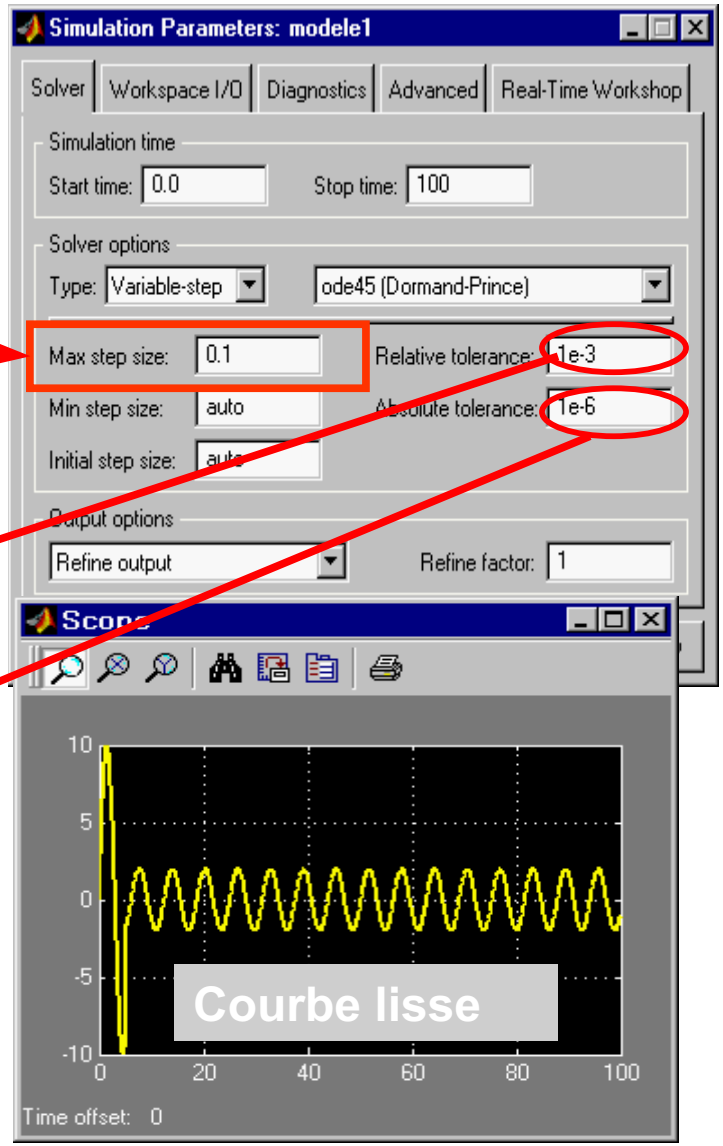
- **Si le Max Step Size est en mode 'auto', Simulink calcule le step size comme suit**

$$h = \frac{t_{stop} - t_{start}}{50}$$

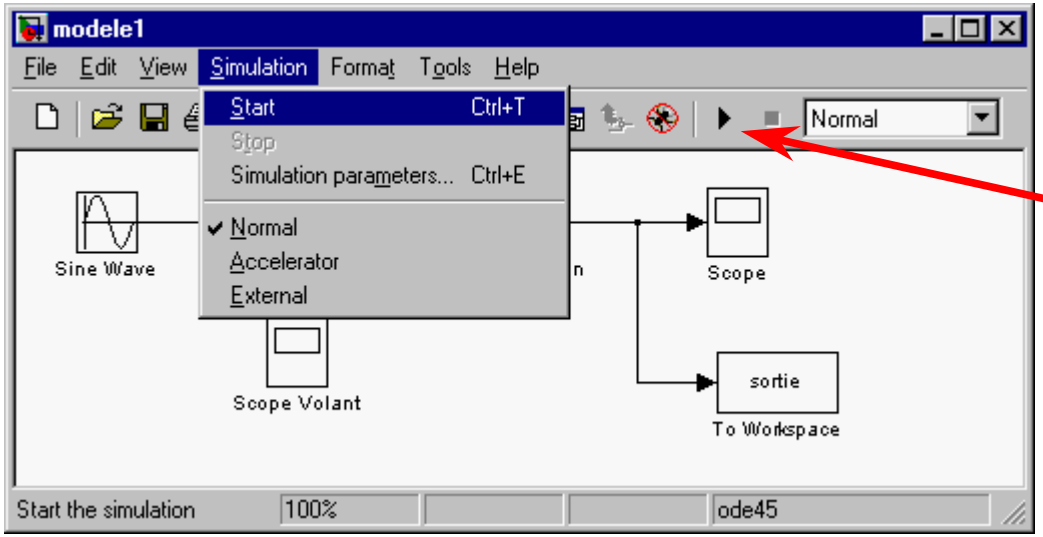
- **Le choix d'un step size petit permet d'avoir une meilleure résolution**

$$\frac{\text{erreur en x}}{\text{x}} < 1e-3$$

$$\text{erreur en x} < 1\text{e-}6$$

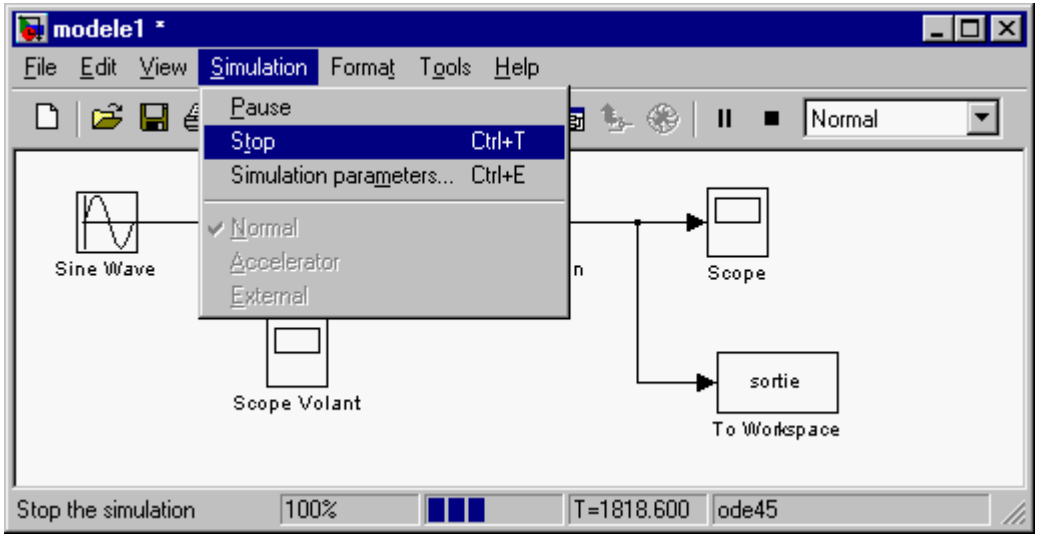


4.2 Gestion de la Simulation



Exécution :

- **menu Simulation/Start**
- **bouton Play**



Arrêt :

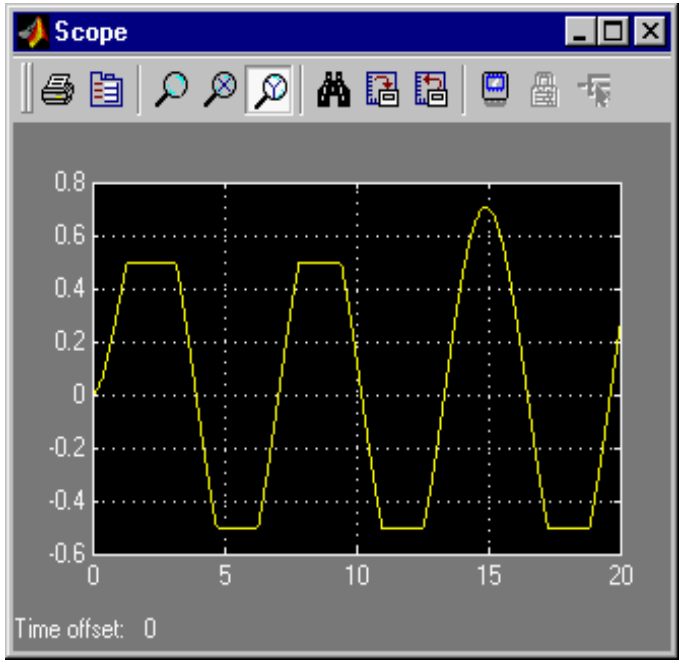
- **menu Simulation/Stop**
- **bouton Stop**
- **automatique lorsque le temps final de simulation est atteint**

4.3 Simulation Intéractive

Pendant la simulation, les paramètres des blocs peuvent être modifiés. Par contre, il est impossible de :

- **ajouter ou d'enlever un bloc**
- **modifier le nombre d'états, d'entrées et de sorties**
- **changer la taille des vecteurs**

Exemple : la borne supérieure de la saturation passe de 0.5 à 1 en cours de simulation.



4.4 Le Scope Volant

Le Scope Volant est utilisé pour visualiser la sortie d'un bloc en cours de simulation sans faire de branchement. Il peut être utilisé de 2 façons :

1.1 Copier le bloc *Scope* avec le bouton droit de la souris et le renommer *Scope Volant*.

1.2 Ouvrir le *Scope Volant* et cocher la case « *floating scope*" dans les propriétés du bloc

1.3 Lancer la simulation

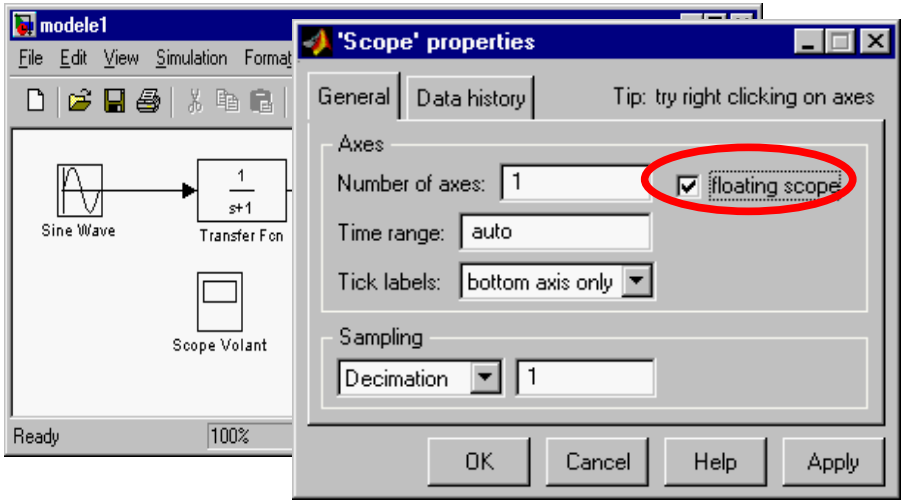
1.4 Cliquer sur l'un des fils : son contenu apparaît dans le *Scope Volant*

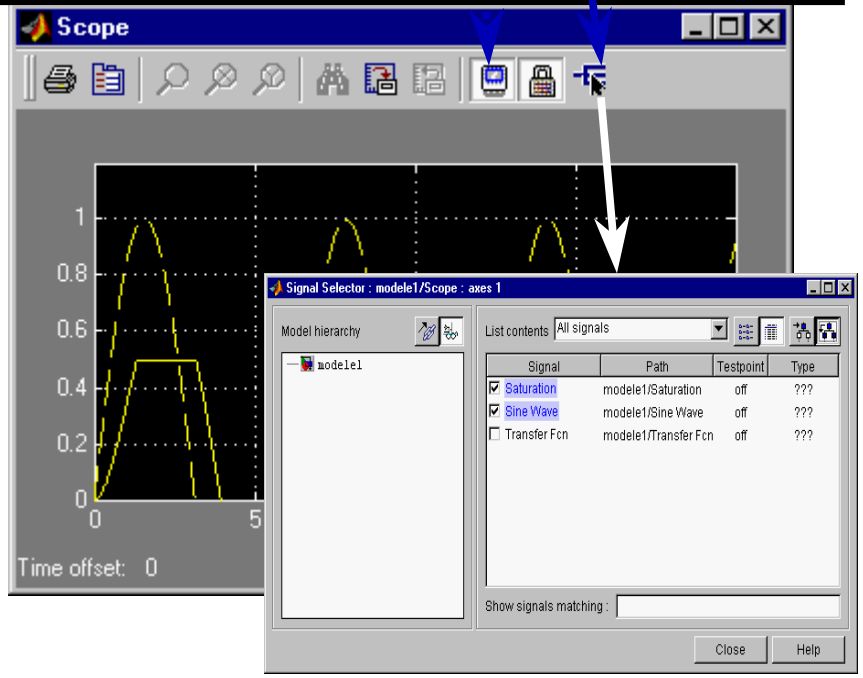
1.5 Cliquer sur un autre fil en tenant appuyé le bouton SHIFT : tous les fils sélectionnés sont visualisés dans le *Scope Volant*.

2.1 Appuyer sur l'icône « *scope volant* » de la barre menu du scope)

2.2 Appuyer sur l'icône « *signal selection* » puis sélectionner les signaux à visualiser.

2.3 Lancer la simulation





Exemples

- **Equation de Mathieu :**

$$\frac{d^2x(t)}{dt^2} = -\omega^2(1+\varepsilon\cos(t))x(t)$$

avec $\varepsilon = 0.01$ **et** $\omega = 0.5$

Construire le schéma-bloc correspondant

Exemples (cont.)

- **Oscillateur**

Construire et comparer les schémas-blocs correspondant aux équations ci-dessous :

$$\frac{d^2x}{dt^2} + 0.3\frac{dx}{dt} + 0.2x = u$$

$$X(s) = \frac{U(s)}{s^2 + 0.3s + 0.2}$$

$$\begin{cases} \dot{x} = \begin{bmatrix} 0 & 1 \\ -0.2 & -0.3 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ y = \begin{bmatrix} 1 & 0 \end{bmatrix} x \end{cases}$$

Exemples (cont.)

Modulateur/Démodulateur

- Créer un signal modulé en amplitude :
  - Utiliser un signal en dent de scie unitaire comme source.
  - Utiliser un signal sinusoïdal d'amplitude 1 et de pulsation 100

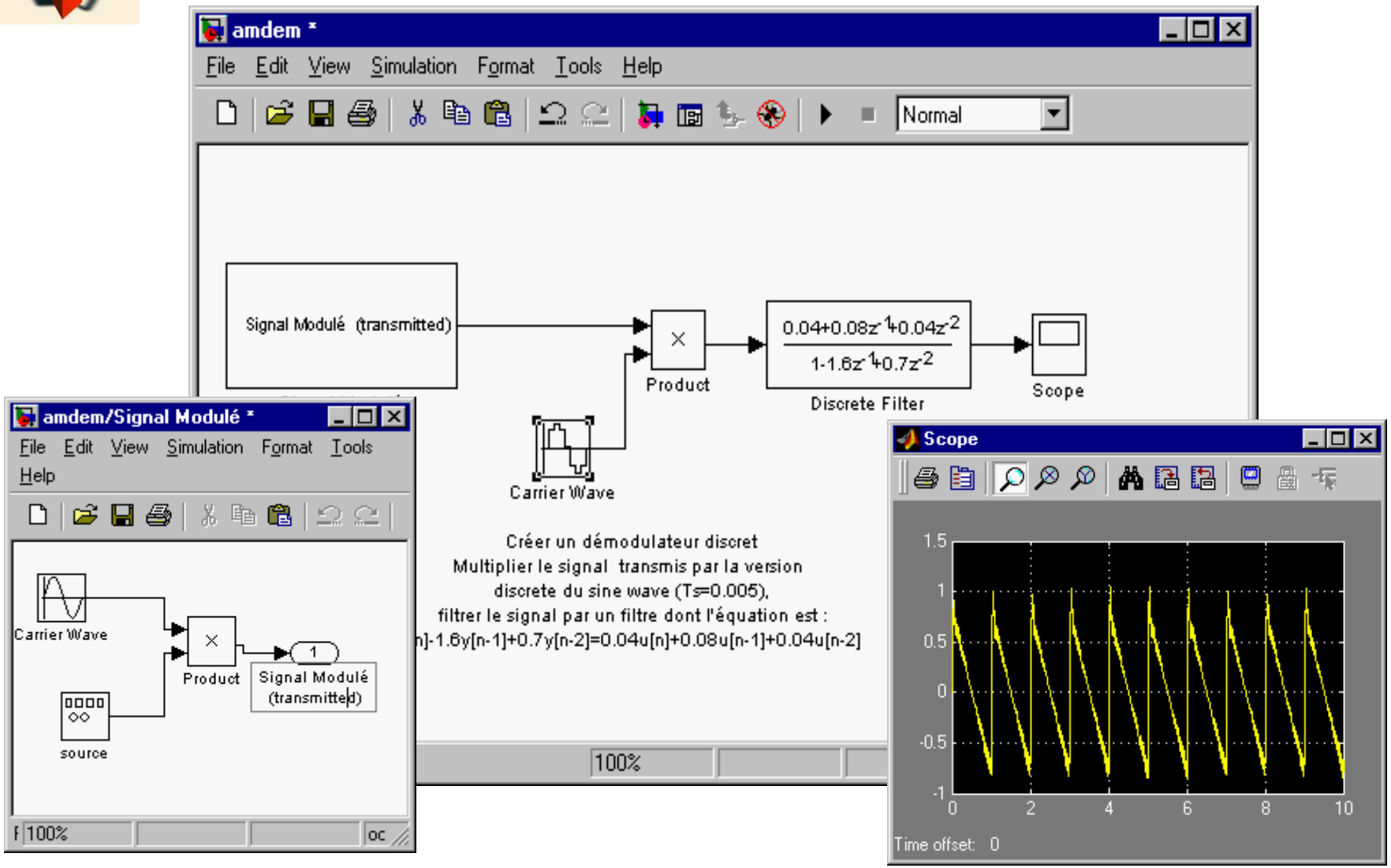
signal modulé =Source x Porteuse

Créer un filtre discret représenté par l'équation suivante :

y(n)-1.6y(n-1)+0.7y(n-2) = 0.04u(n)+0.08u(n-1)+0.04u(n-2)

- Utliser le signal modulé comme source.
- Multiplier le signal transmis par une forme discrete de la porteuse (frequency=100, sample time=5 ms)
- Filtrer le signal résultant avec le filtre ci-dessus.
- Observer le signal transmis ainsi que la sortie du filtre.

Solution


amdem *
File Edit View Simulation Format Tools Help
Normal
Signal Modulé (transmitted)
Product
0.04+0.08z-1+0.04z-2
1-1.6z-1+0.7z-2
Discrete Filter
Scope
Carrier Wave
Créer un démodulateur discret
Multiplier le signal transmis par la version
discrete du sine wave (Ts=0.005),
filtrer le signal par un filtre dont l'équation est :
n]-1.6y[n-1]+0.7y[n-2]=0.04u[n]+0.08u[n-1]+0.04u[n-2]
100%
amdem/Signal Modulé *
File Edit View Simulation Format Tools
Help
Carrier Wave
Product
1
Signal Modulé
(transmitted)
source
100%
oc
Scope
1.5
1
0.5
0
-0.5
-1
0
2
4
6
8
10
Time offset: 0

Exemples (Cont.)

Canal de communication

- Implanter la dynamique d'un canal de communication :

  Utiliser

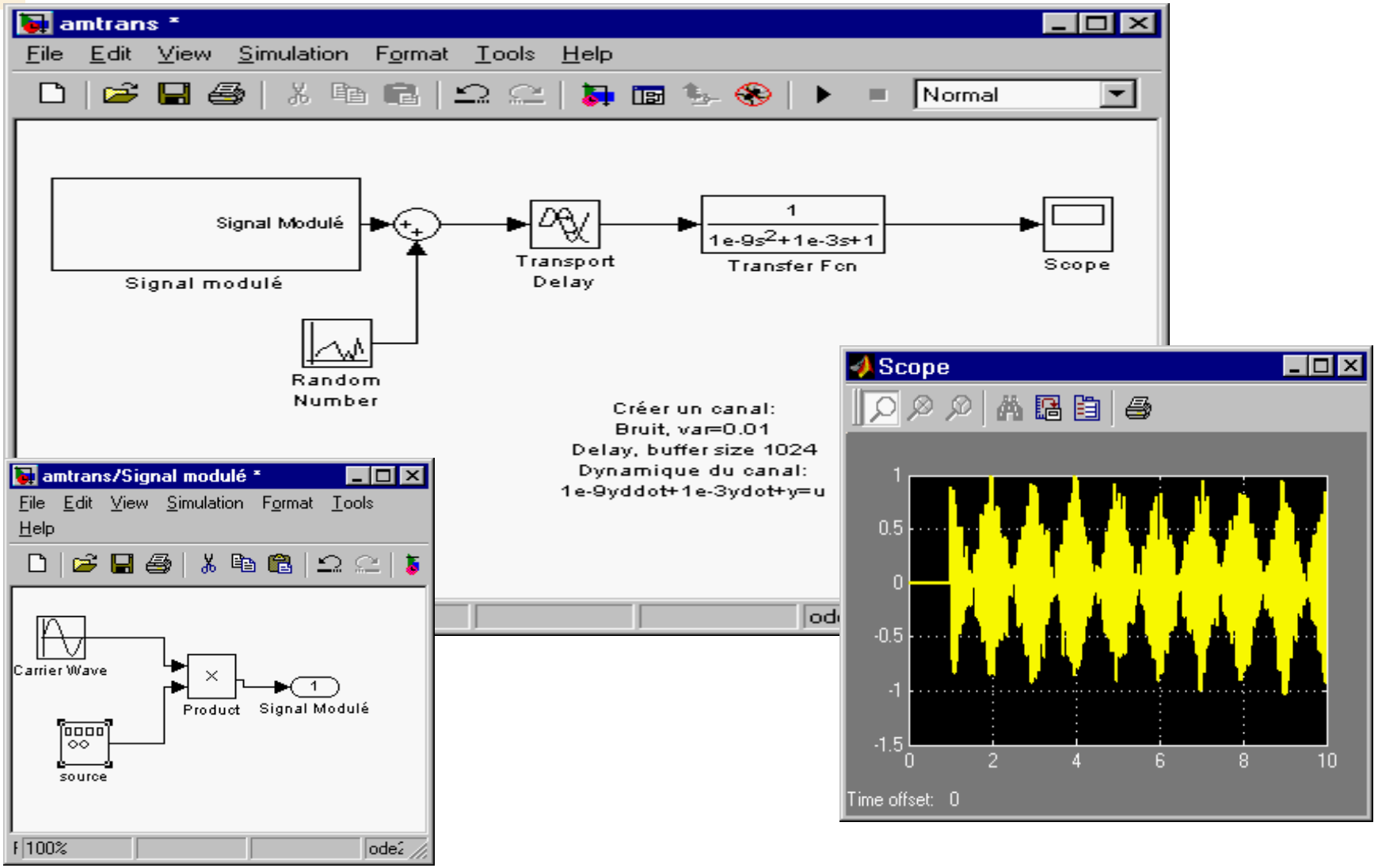
  - un bruit Additive "noise corruption" avec une variance de 0.01
  - un bloc delay avec une taille buffer de 1024
  - l'équation décrivant la dynamique du canal

$$10^{-9}\ddot{y} + 10^{-3}\dot{y} + y = u$$

  Utiliser le signal modulé de l'exemple précédent.

- Observer la sortie
- Essayer differents solvers

Solution:


amtrans *
File Edit View Simulation Format Tools Help
Normal
Signal Modulé
Signal modulé
Transport Delay
1
1e-9s2+1e-3s+1
Transfer Fcn
Scope
Random Number
Créer un canal:
Bruit, var=0.01
Delay, buffer size 1024
Dynamique du canal:
1e-9yddot+1e-3ydot+y=u
amtrans/Signal modulé *
File Edit View Simulation Format Tools Help
Carrier Wave
Product
1
Signal Modulé
source
100%
Scope
1
0.5
0
-0.5
-1
-1.5
0
2
4
6
8
10
Time offset: 0

4.5 Les Types de Données

- **Les tailles des signaux**

Les tableaux 2D [mxn]

Les vecteurs lignes 2D [1xn]

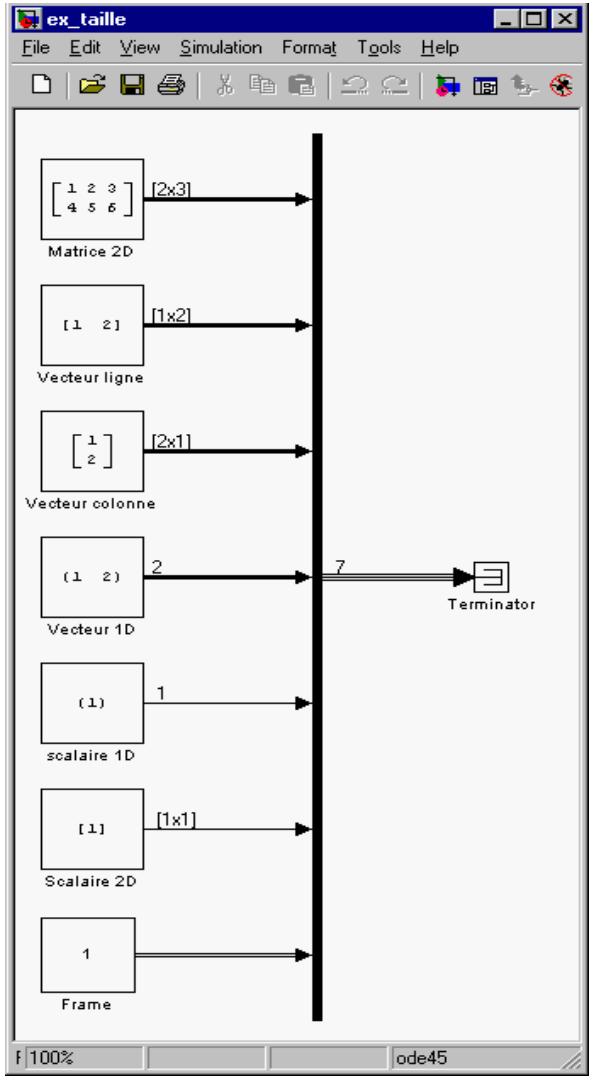
Les vecteurs colonnes 2D [mx1]

Les vecteurs 1D [n]

Les scalaires 1D [1]

Les scalaires 2D [1x1]

Les « Frames » [mx1]



4.5 Les Types de Données (cont.)

- **Les différents types de données**

SIMULINK travaille par défaut en double précision. Il est possible de choisir le type de données qui sont véhiculées et traitées dans le schéma-blocs. Ce typage permet de réduire le stockage et de travailler avec des types de données appropriés.

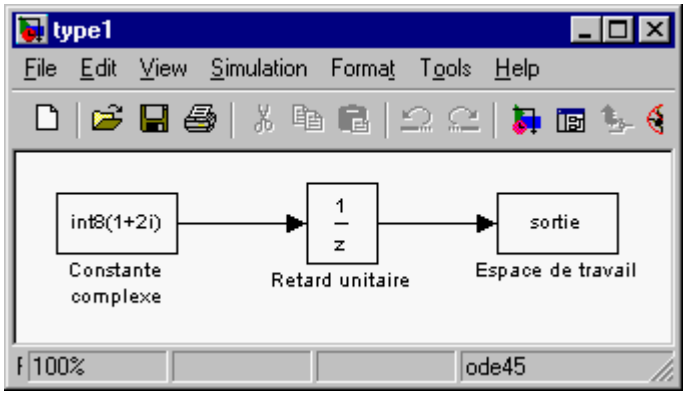
Voici les différents types de données disponibles :

- boolean 1 octet,
- int8, uint8 1 octet,
- int16, uint16 2 octets,
- int32, uint32 4 octets,
- single 4 octets,
- double 8 octets.

4.5 Les Types de Données (cont.)

- **Les différents types de données (cont.)**

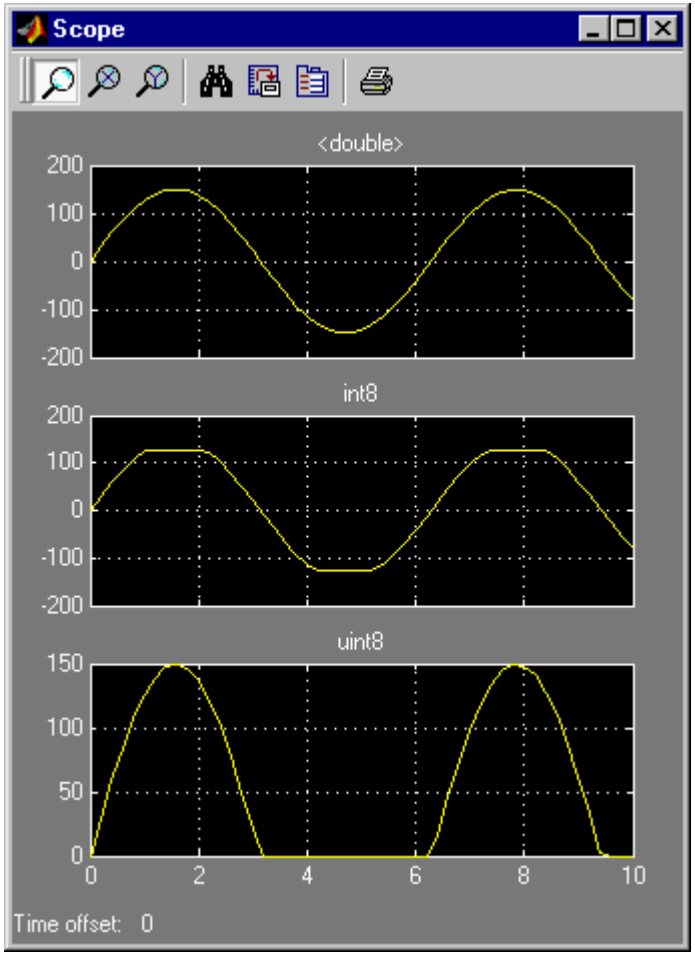
SIMULINK supporte le type de données complexes.

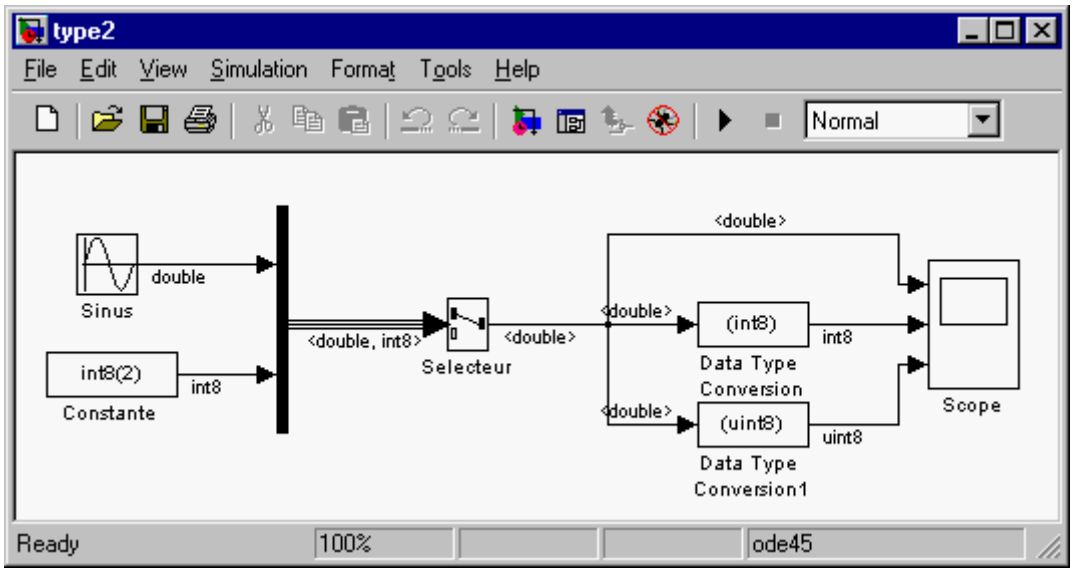



```
>> whos
  Name          Size              Bytes  Class

  sortie       11x1                 22  int8 array
(complex)
  tout         11x1                 88  double array

Grand total is 22 elements using 110 bytes
```

4.5 Les Types de Données (cont.)

- **Les différents types de données (cont.)**





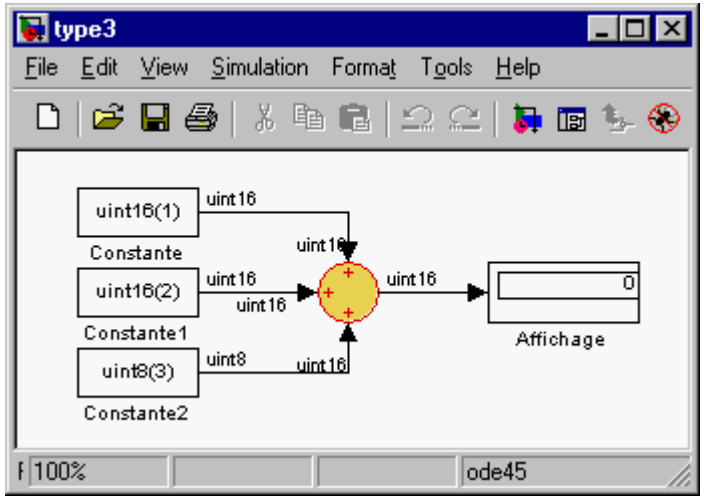
• Dans un même schéma SIMULINK, plusieurs types peuvent cohabiter. Il est, par exemple, possible de mixer des types différents sur un même lien ou de les convertir.

4.5 Les Types de Données (cont.)

- **Opérations sur les différents types**

Pour les blocs :

- *Sum,*
- *Product,*
- *Relational Operator,*
- *Merge,*
- *Switch,*
- *Manuel Switch,*
- *Multiport Switch,*
- *Logical Operator,*



tous les types en entrée doivent être identiques et le type de données en sortie est le même que celui en entrée du bloc.

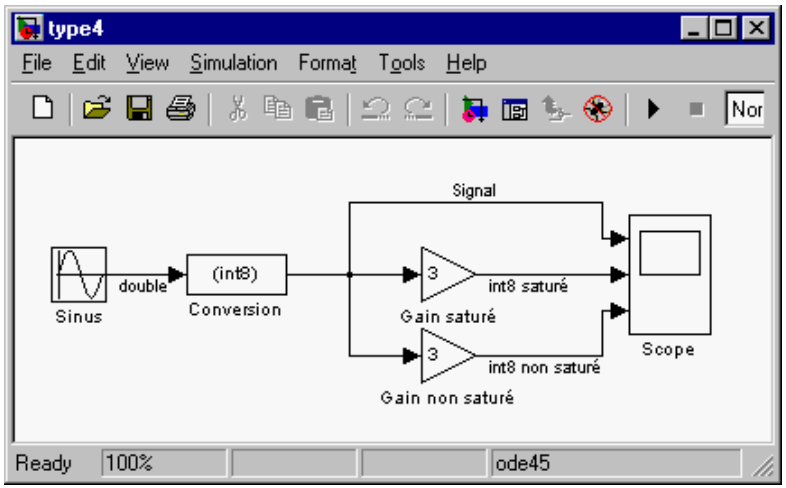
4.5 Les Types de Données (cont.)

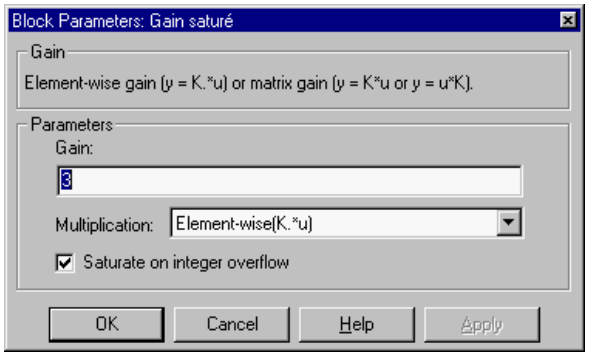
- **Opérations sur les différents types (cont.)**

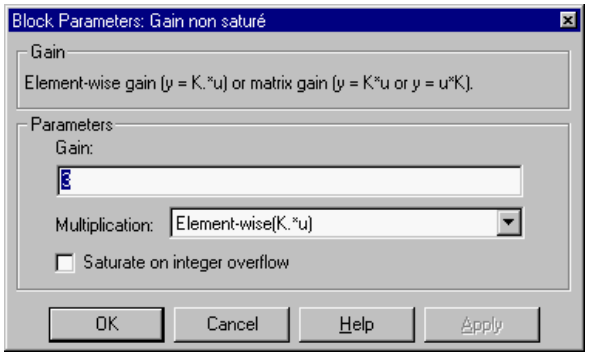
Les blocs :

- *Sum*
- *Product*
- *Gain*
- *Data type Conversion*

permettent de gérer le dépassement du type entier.

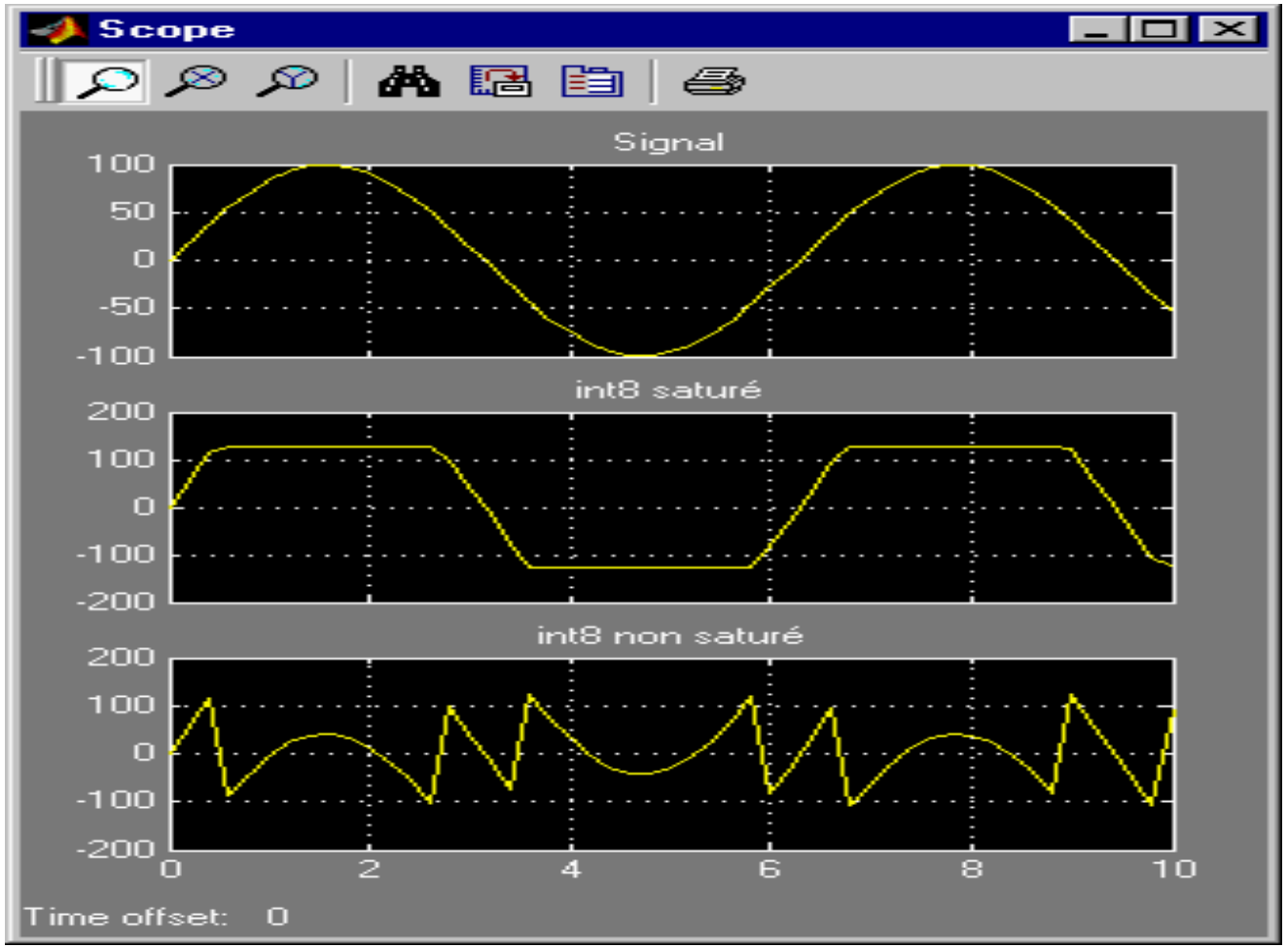






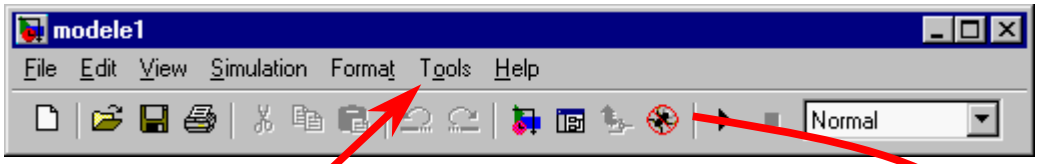
4.5 Les Types de Données (cont.)

- **Opérations sur les différents types (cont.)**

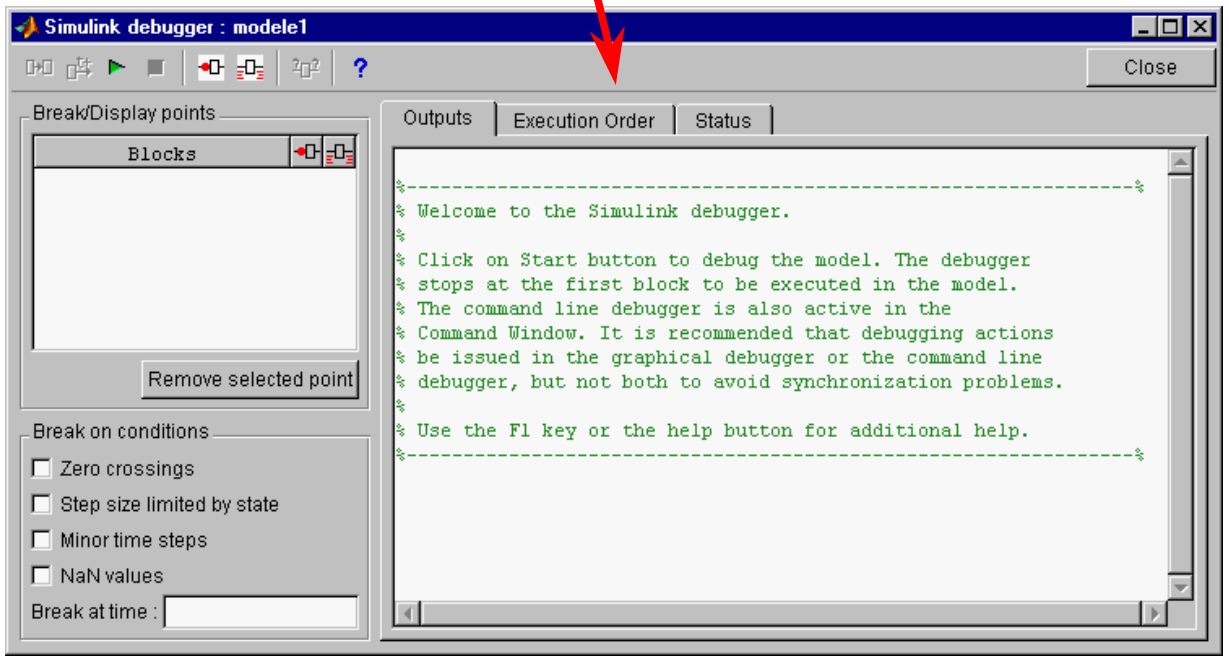


4.6 Debugger

- **Le Debugger Graphique**



Tools/Simulink Debugger



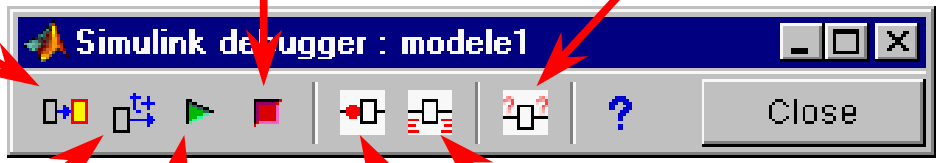
4.6 Debugger (cont.)

- **Le Debugger Graphique (cont)**

Exécution des blocs pas-à-pas pour un temps de simulation fixé

Arrêt du mode debugger

Affichage des entrées/sorties du bloc sélectionné même si celui ci n'est pas en cours d'évaluation



Exécution d'un pas d 'intégration complet

Affichage des entrées/sorties du bloc sélectionné à chaque fois qu'il est évalué

Lancement de la simulation en mode debugger - Exécution en continu de points d 'arrêt en points d 'arrêt

Mise en place des points d'arrêt devant le bloc sélectionné

4.6 Debugger (cont.)

- **Le Debugger en Ligne**

Pour lancer la simulation en mode debugger :

```
>> sldebug 'modele'
```

Commandes de gestion de la simulation

`step`	Exécution des blocs pas à pas
`next`	Exécution d'un pas de temps complet
`continue`	Exécution jusqu'au prochain point d'arrêt
`run`	Exécution jusqu'à la fin de la simulation
`quit`	Quitte le mode debugger

Commandes de visualisation en cours de debuggage

`slist`	Liste des blocs dans leur ordre d'exécution
`systems`	Liste des systèmes concernés
`break gcb` ou `break s:b`	Mise en place des points d'arrêt
`disp gcb` ou `disp s:b`	Affichage des grandeurs d'un bloc
`states`	Affichage des états du modèle
`status`	Etat de la simulation

4.6 Debugger (cont.)

- **Le Debugger en Ligne (cont.)**

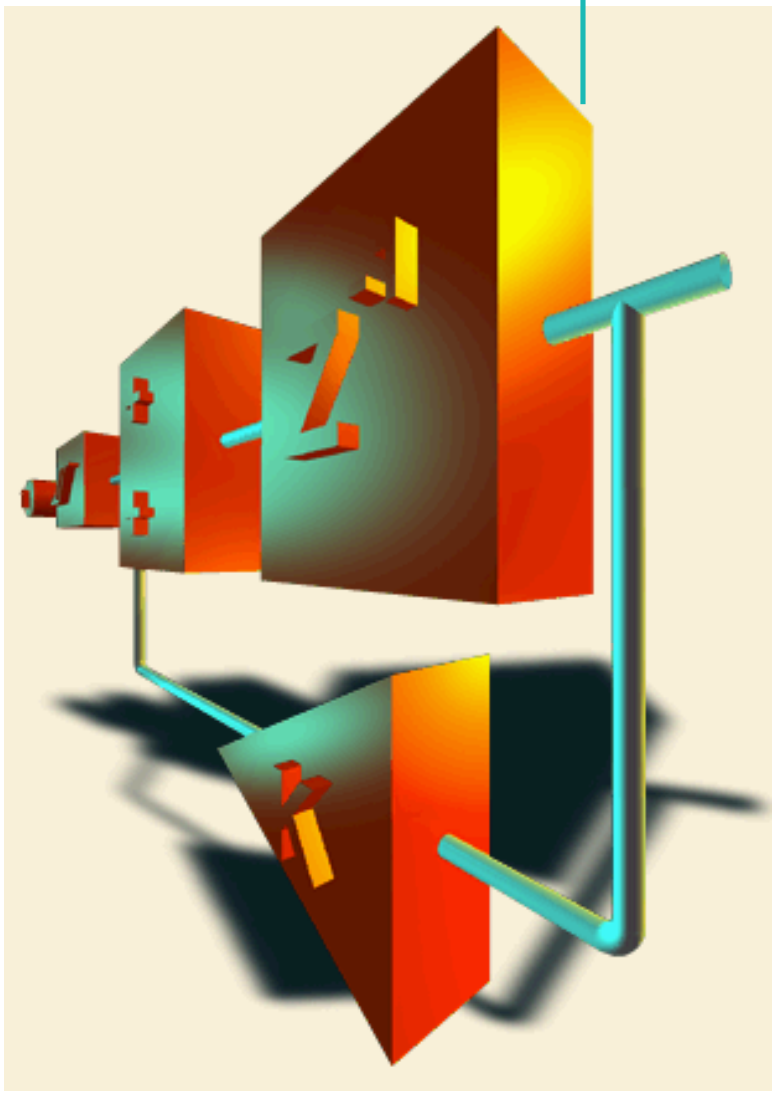
```
>> sldebug 'modele1'
[Tm=0                        ] **Start** of system 'modele1' outputs

(sldebug @0:0 'modele1/Transfer Fcn'): slist
---- Sorted list for 'modele1' [6 blocks, 6 nonvirtual blocks, directFeed=0]
  0:0    'modele1/Transfer Fcn' (TransferFcn)
  0:1    'modele1/Saturation' (Saturate)
  0:2    'modele1/Scope' (Scope)
  0:3    'modele1/Sine Wave' (Sin)
  0:4    'modele1/To Workspace' (ToWorkspace)
  0:5    'modele1/Scope Volant' (Scope)

(sldebug @0:0 'modele1/Transfer Fcn'): next
[Tm=0.2                      ] **Start** of system 'modele1' outputs
```

4.6 Debugger (cont.)

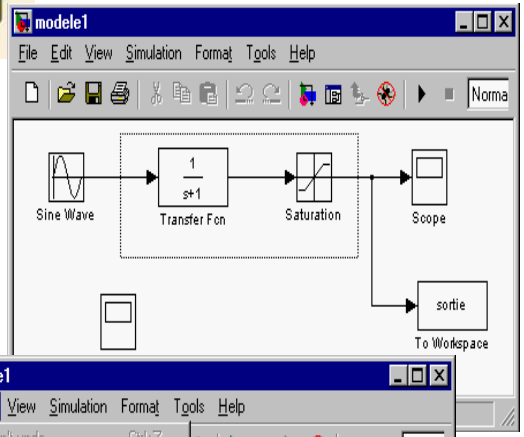
- **Le Debugger en Ligne (cont.)**

```
(sldebug @0:0 'modele1/Transfer Fcn'): next
[Tm=0.2                    ] **Start** of system 'modele1' outputs

(sldebug @0:0 'modele1/Transfer Fcn'): states
Continuous state vector (value,index,name):
  0.01866677260531322        0 (0:0 'modele1/Transfer Fcn')

(sldebug @0:0 'modele1/Transfer Fcn'): disp 0:1
Display of block 0:1 'modele1/Saturation' installed.

(sldebug @0:0 'modele1/Transfer Fcn'): next
[Tm=0.4                    ] I/O of block 0:1 'modele1/Saturation'
 U1 = [0.06933872878056514]
 Y1 = [0.06933872878056514]
[Tm=0.4                    ] **Start** of system 'modele1' outputs
```

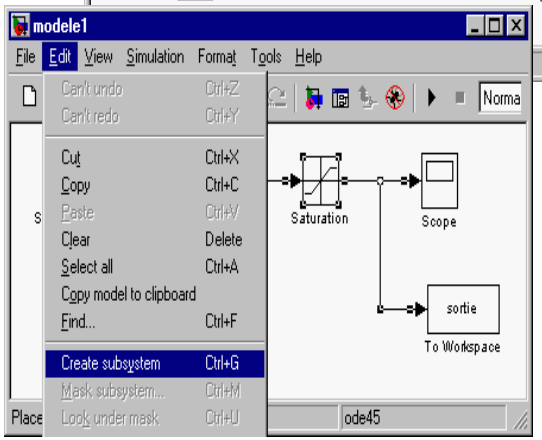
5. Construction de Blocs



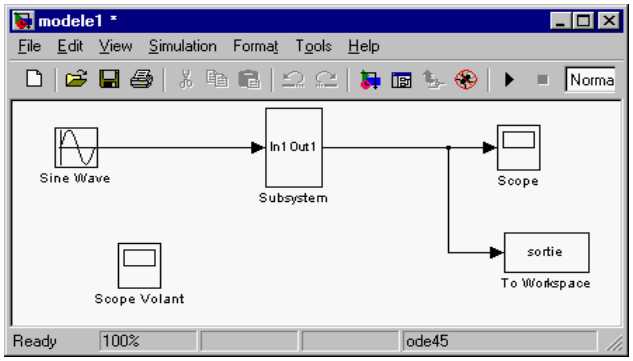
5.1 Construction de Sous-Systèmes



• Sélectionner les blocs à regrouper avec la souris (liens d'entrée et de sortie inclus)



• Grouper avec le menu **Edit/Create Subsystem**



Maintenant, un nouveau bloc remplace tous les blocs qui étaient sélectionnés. Le nom de ce nouveau bloc est par défaut *Subsystem*

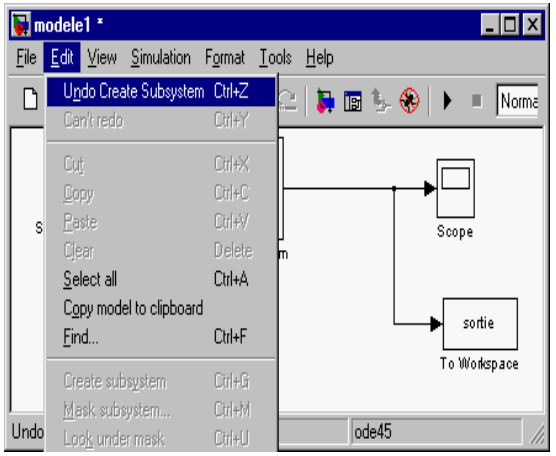
• Il est possible d'annuler (Undo) les effets jusqu'à 101 exécutions consécutives par le choix menu **Edit/Undo**. Les opérations pouvant être annulées sont :

• Ajouter, effacer, ou déplacer un bloc, une ligne ou une annotation

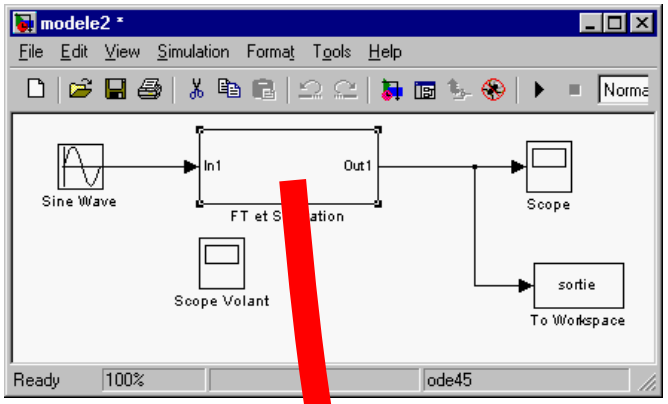
• Éditer un nom de bloc

• Créer un sous-système

• Pour dégrouper sélectionner le menu **Edit/Undo Create Subsystem**

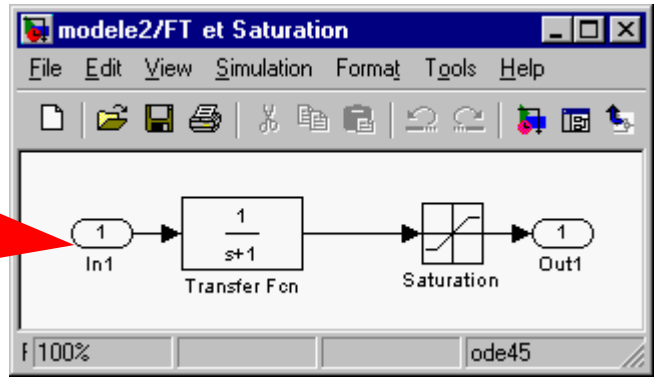


5.1 Construction de Sous-Systèmes



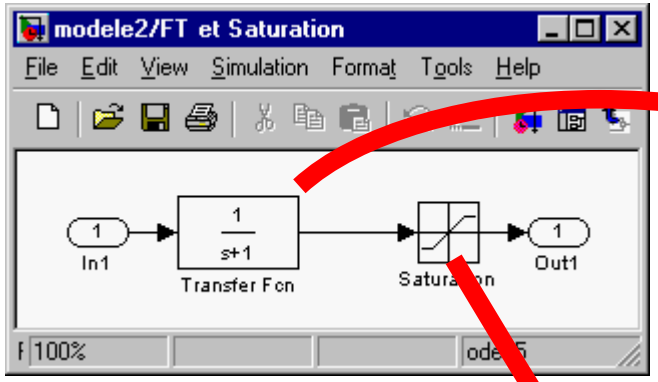
- Enregistrer sous modele2.mdl et renommer le bloc *Subsystem* par *FT et Saturation*.

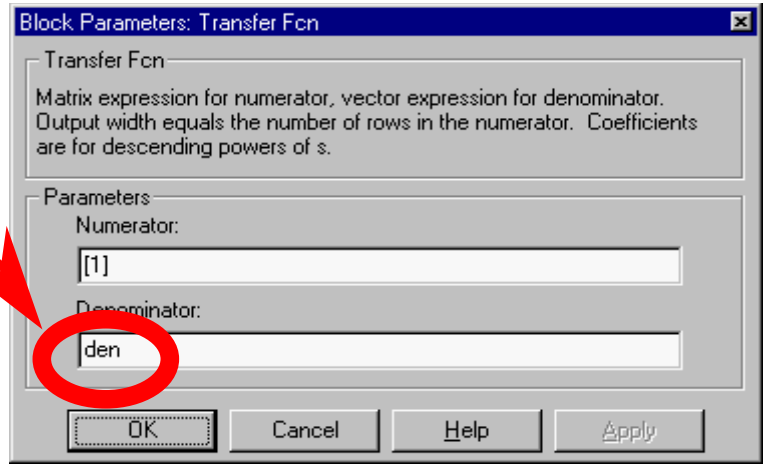
Double-clic

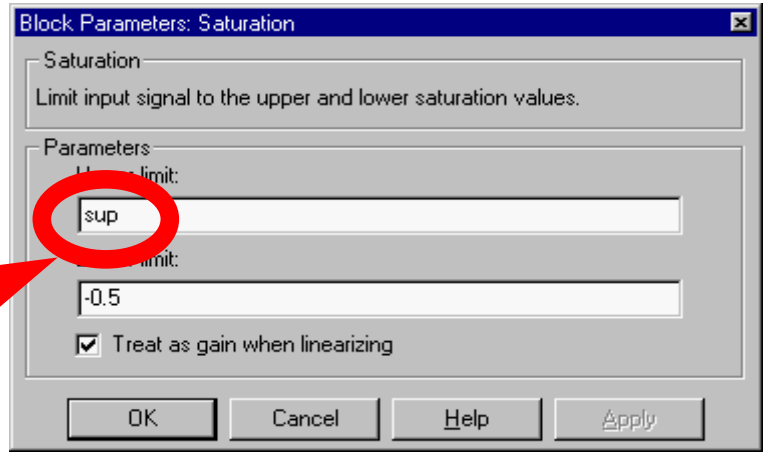


5.2 Paramétrage

Les blocs peuvent être paramétrés : les valeurs sont remplacées par des variables MATLAB. Ces variables doivent être définies dans MATLAB avant le lancement de la simulation.







Les variables *den* et *sup* sont définies dans MATLAB :

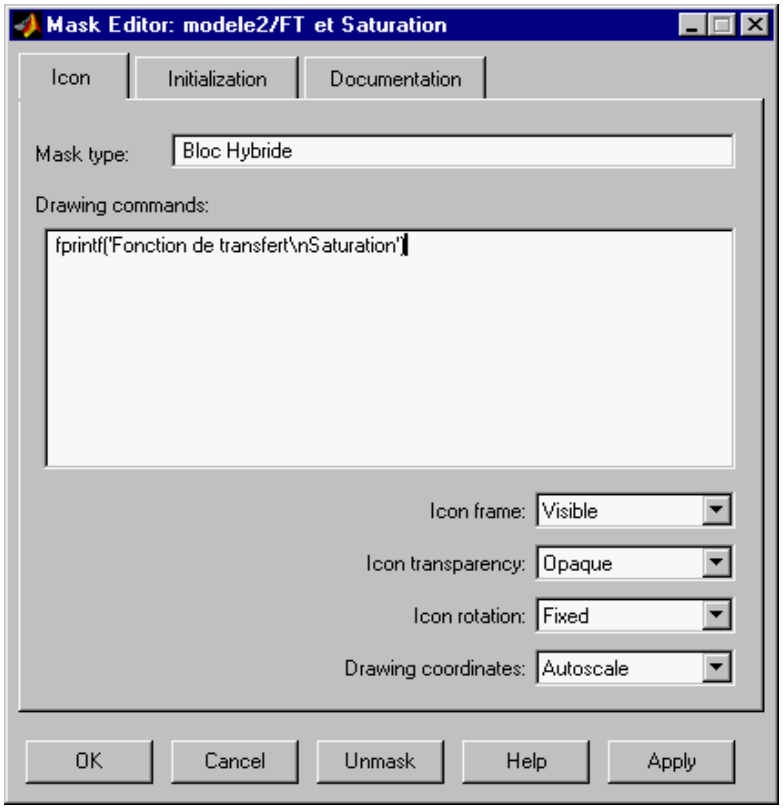
```
>> den = [1 1];
>> sup = 0.5;
```

5.3 Le Masquage

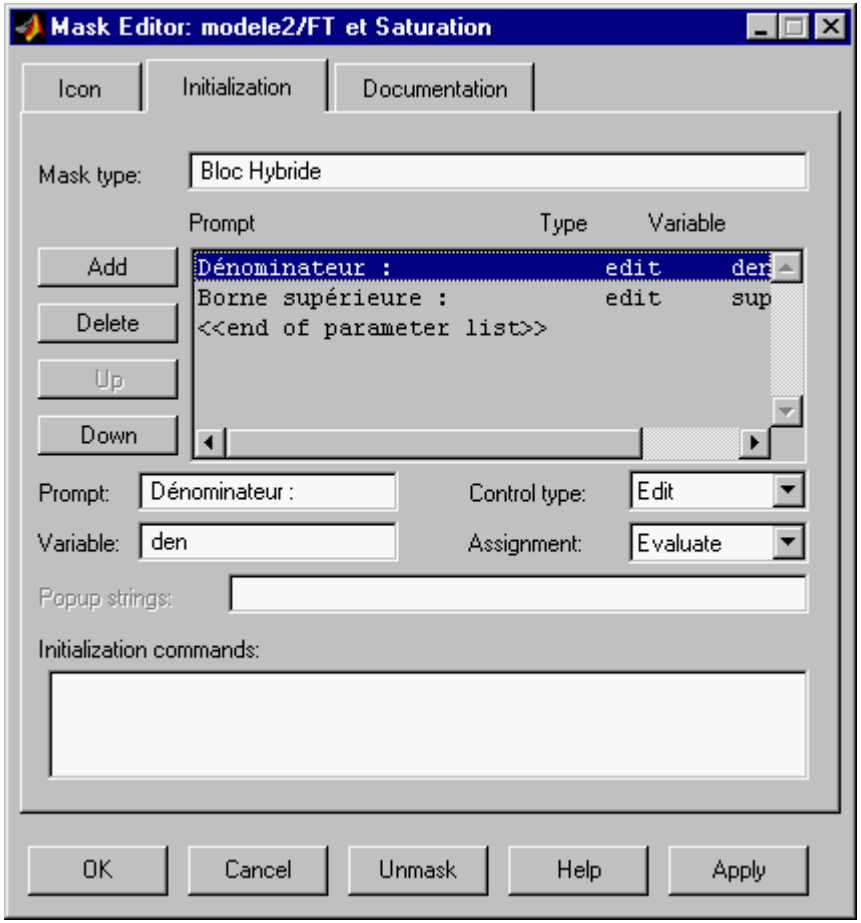
Le masquage permet de :

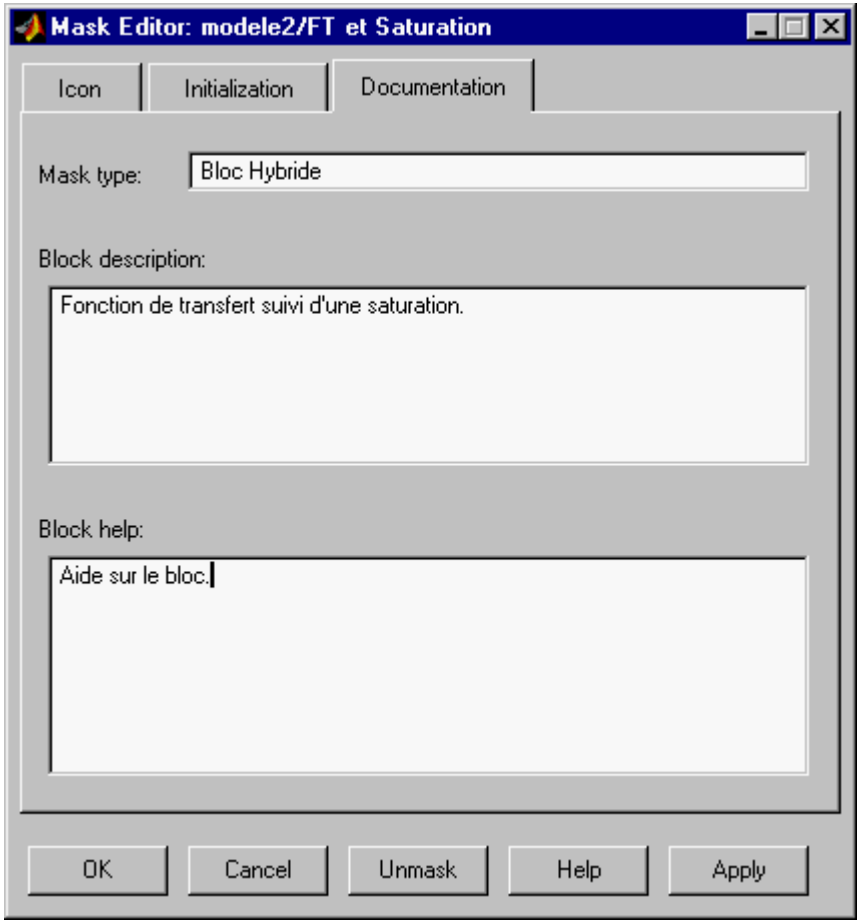
- Changer les caractéristiques d'un bloc. Par exemple, un bloc *Zero-Pole* peut être transformé en filtre de Butterworth
- Personnaliser et simplifier des modèles

• Sélectionner le bloc *FT et Saturation* en cliquant dessus.
• Masquer ce bloc avec le menu Edit/Mask Subsystem et remplir les champs.



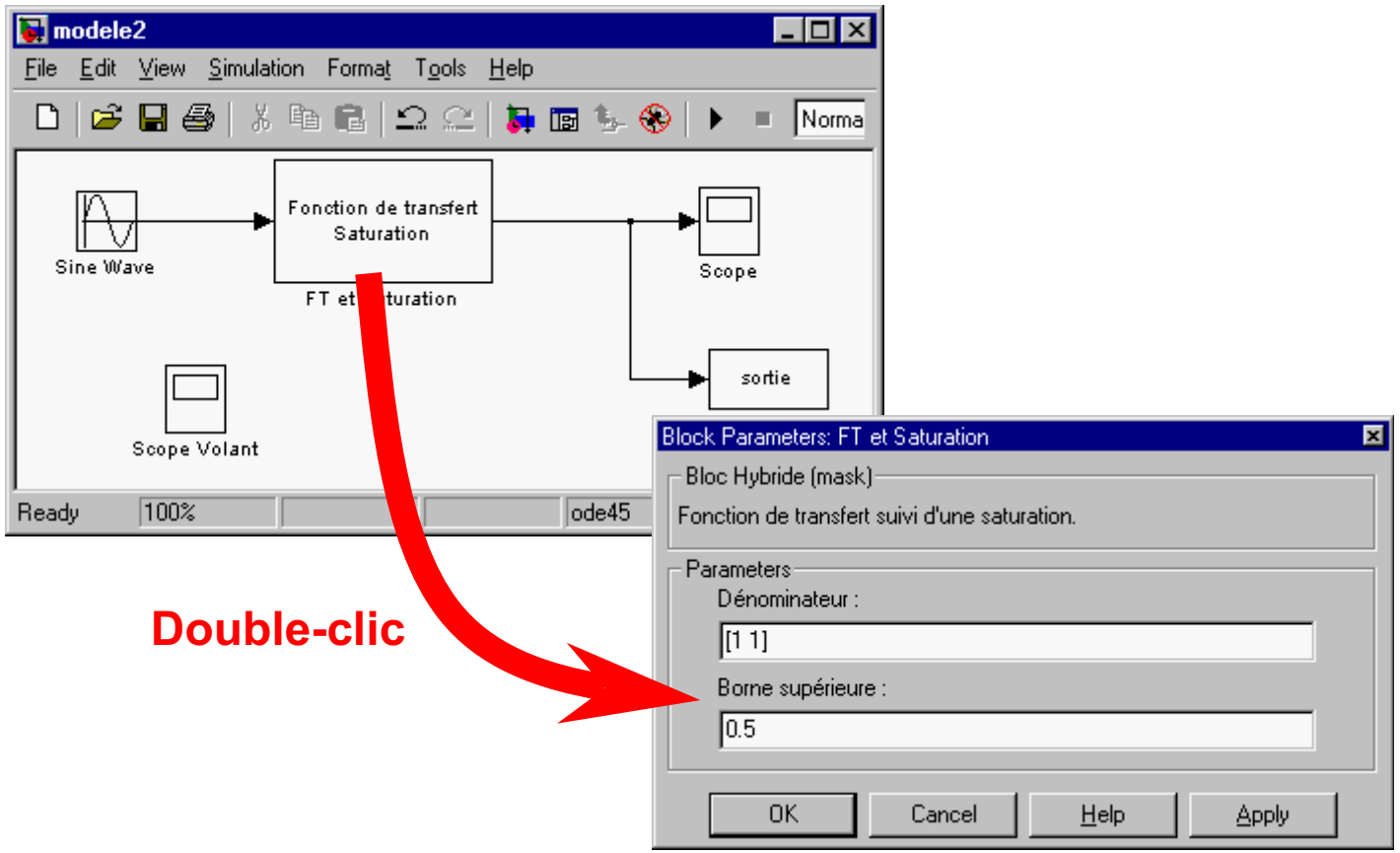
5.3 Le Masquage (cont.)





5.3 Le Masquage (cont.)

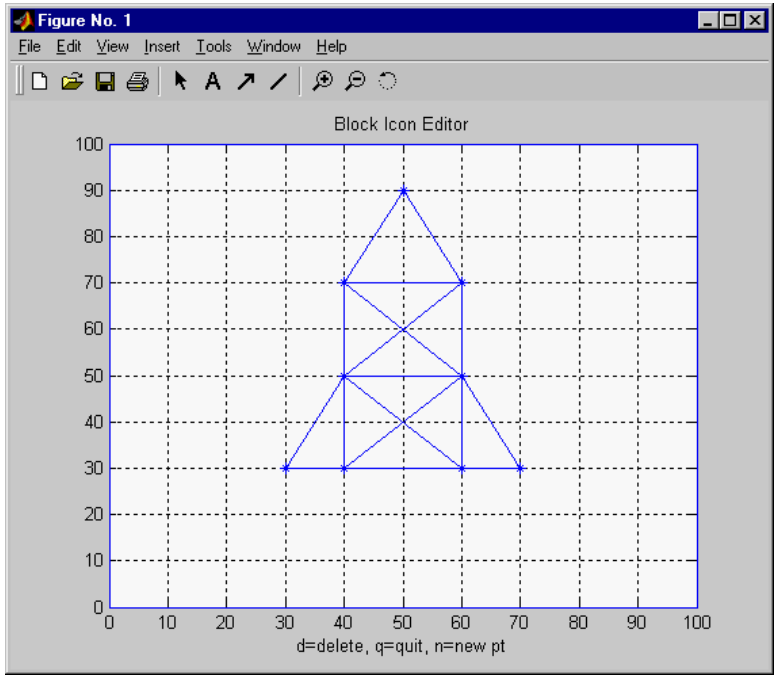
Le résultat du masquage est le suivant :

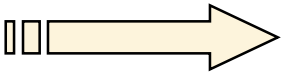


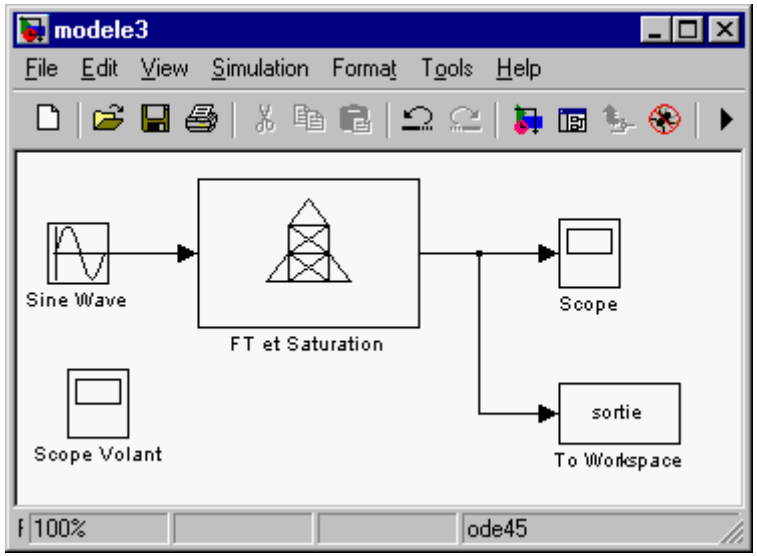
5.3 Le Masquage (cont.)

La commande « **iconedit** » permet de construire une icône graphique : le champ du masque "*Drawing commands*" est complété automatiquement avec la commande plot :

```
>> iconedit
Name of block window: modele2      ⇒ nom du fichier
Name of block: FT et Saturation    ⇒ nom au-dessous du bloc (respecter
                                     les majuscules, minuscules, et blancs)
```







5.4 Bibliothèques

- **Vocabulaire**

Library : ensemble de blocs. Une bibliothèque est créée uniquement avec le menu **File/New/Library** d'une fenêtre SIMULINK.

bloc-library *:* bloc dans une bibliothèque.

bloc-référence *:* copie d'un *bloc-library*. Un *bloc-référence* est lié à un *bloc-library*. Une modification du *bloc-library* entraîne automatiquement une modification du *bloc-référence*.

Lien : connexion entre le *bloc-référence* et son *bloc-library*. Le lien permet à une modification du *bloc-library* d'entraîner automatiquement une modification du *bloc-référence.*

Copie *:* opération qui crée un *bloc-référence* à partir soit d'un *bloc-library*, soit d'un autre *bloc-référence*.

5.4 Bibliothèques (cont.)

• Créer une Bibliothèque

• Créer une bibliothèque avec le menu **File/New/Library**.
• Placer le bloc *FT et Saturation*, et sauvegarder la bibliothèque sous le nom : *biblio_perso.mdl*.

• Modifier une Bibliothèque

• Lorsqu'une bibliothèque est ouverte (par exemple `>>biblio_perso`), elle est automatiquement verrouillée. Les blocs ne sont pas modifiables.

• Pour déverrouiller la bibliothèque, et ainsi pouvoir modifier son contenu, utiliser le menu **Edit/Unlock Library** de la bibliothèque. Elle sera de nouveau verrouillée automatiquement à sa fermeture.

5.4 Bibliothèques (cont.)

- **Copier un *bloc-library* dans un modèle**

  En copiant un *bloc-library* dans un modèle, un lien est créé entre le *bloc-library* et le *bloc-référence*. Les paramètres du bloc sont modifiables, mais pas son masque, ni le contenu du sous-système s'il existe.

- **Supprimer le lien vers le *bloc-library***

  Le lien entre le *bloc-référence* et le *bloc-library* peut être supprimé :

  - pour modifier la bibliothèque sans influencer le modèle,
  - pour avoir un modèle autonome, sans appel à la bibliothèque.

  La suppression du lien est faite avec le menu **Edit/Break Library Link** du modèle.

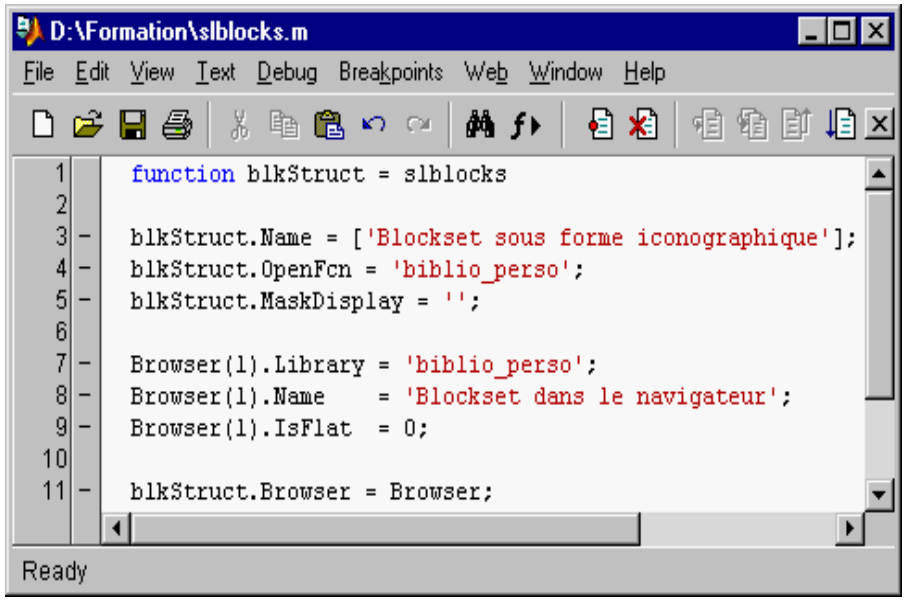
- **Rechercher la bibliothèque**

  Pour retrouver le *bloc-library* auquel est lié le *bloc-référence*, utiliser le menu **Edit/Go To Library Link** du modèle. Alors, la bibliothèque est ouverte avec le *bloc-library* déjà sélectionné.

5.4 Bibliothèques (cont.)

- **Référencer sa propre bibliothèque dans l'interface Simulink**

• Pour utiliser sa propre bibliothèque au même titre que les bibliothèques de base de SIMULINK, il faut la référencer. Pour cela, il faut ajouter, dans le répertoire où se trouve la bibliothèque, le fichier **slblocks.m** que l'on aura personnalisé.



• Pour que ceci soit pris en compte au démarrage, il est nécessaire que le chemin de **slblocks.m** soit connu : soit référencé dans le path MATLAB, soit dans le répertoire courant.

• Pour la création du fichier slblocks.m, un fichier template est fourni dans :
`$matlabroot\toolbox\simulink\blocks\slblocks.m`

5.5 Lisibilité

- **Noms des fils**

Pour clarifier un modèle de taille importante, les fils peuvent porter un nom. Ce nom est propagé dans le modèle lorsqu'il passe à travers les blocs suivants :

Demux, *Enable*, *From*, *Inport*, *Mux*, *Selector*, *Subsystem*.

Pour associer un nom à un fil :

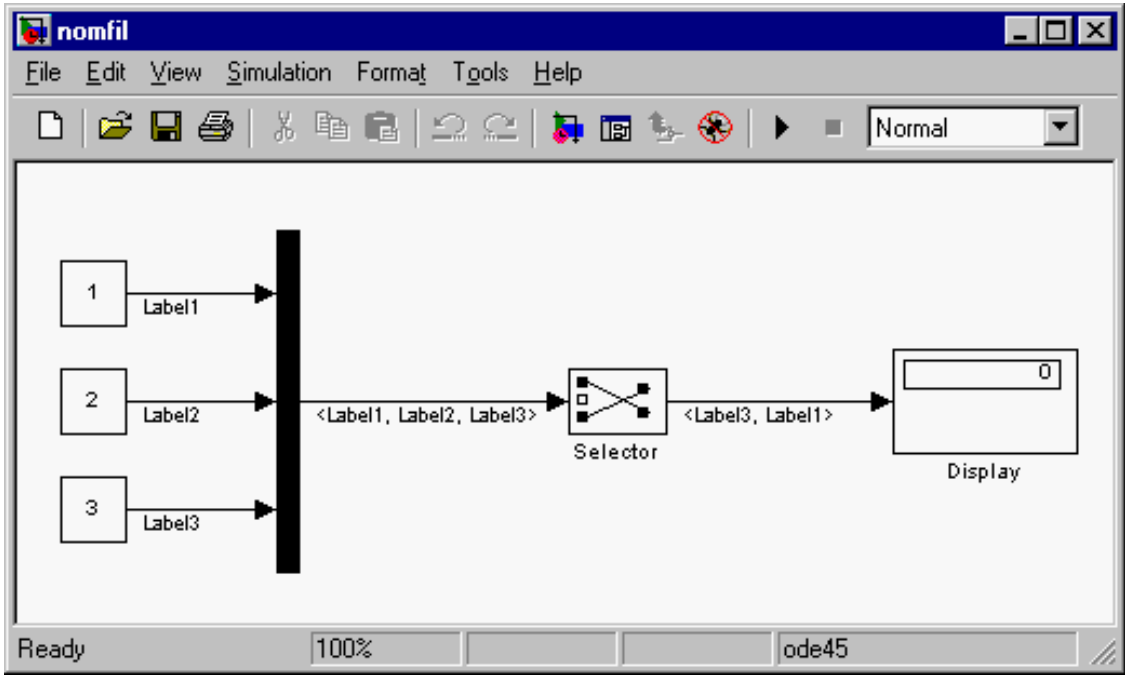
- double-cliquer sur le fil
- saisir le nom

Pour obtenir le nom d'un fil, si celui-ci a déjà été défini :

- double_cliquer sur le fil
- saisir le caractère **<** (inférieur)
- mettre à jour le modèle avec le menu **Edit/Update Diagram**

5.5 Lisibilité (cont.)

- **Noms des fils (cont.)**

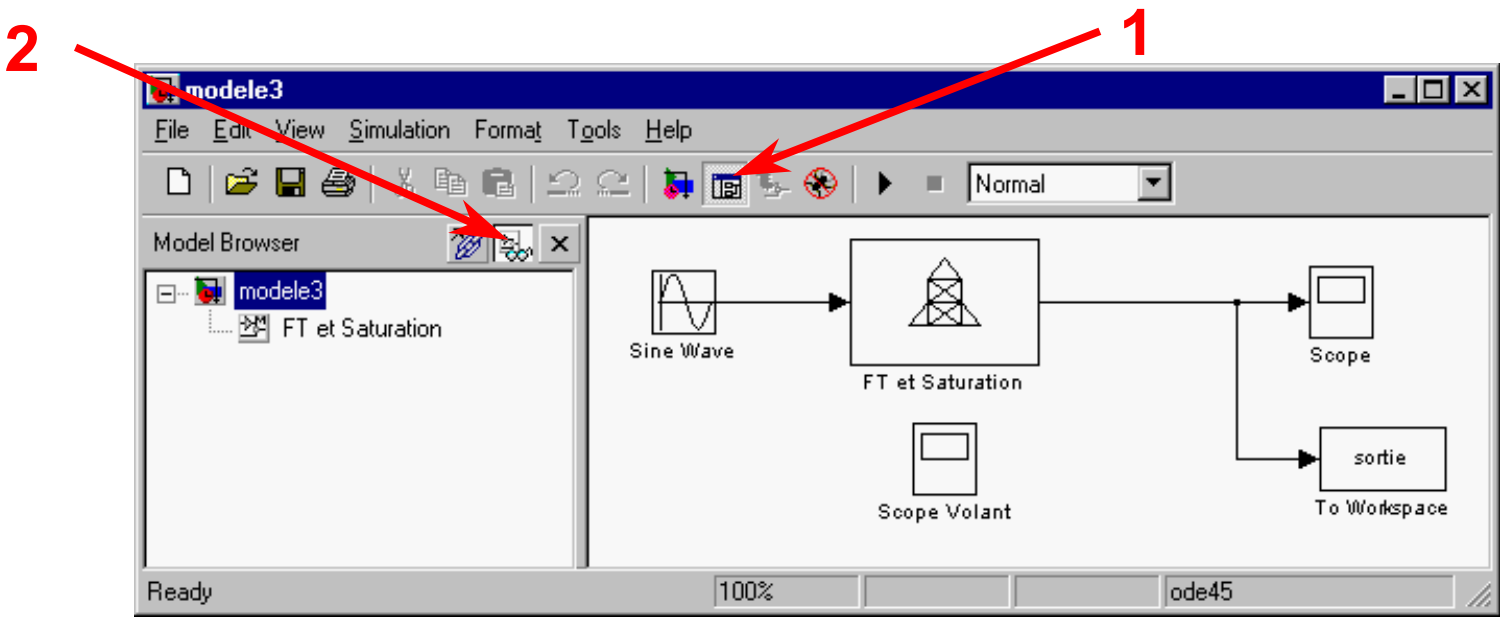


5.5 Lisibilité (cont.)

• Navigateur du modèle

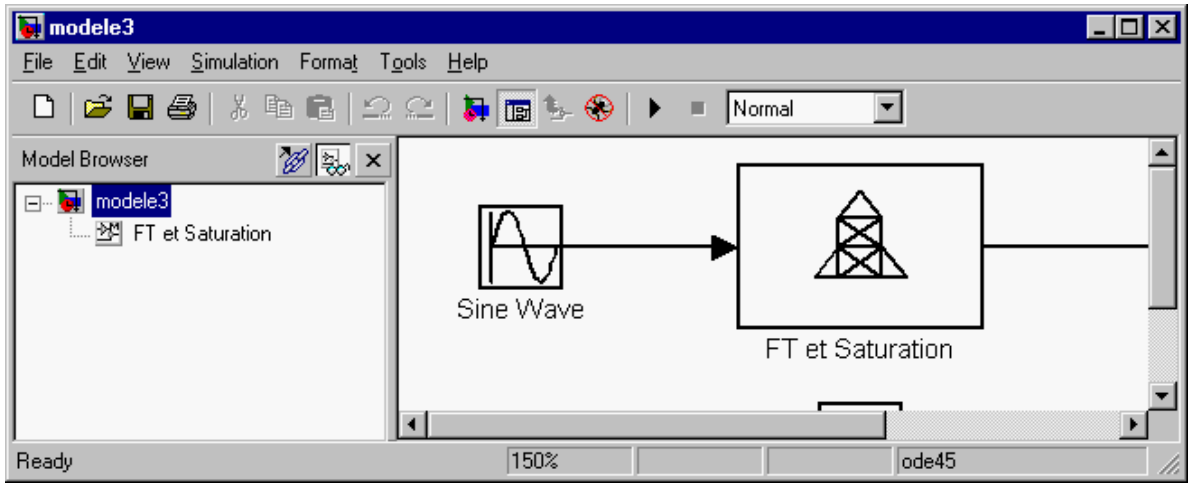
• Le navigateur du modèle, à l'identique du navigateur SIMULINK, permet de se déplacer dans l'arborescence du modèle parmi les sous-systèmes.

Pour activer cette option, il faut cliquer sur **View/Model Browser Options/Model Browser** puis sur **View/Model Browser Options/Look Under Masks** si certains blocs sont masqués. L'autre possibilité est de cliquer sur l'icône **Toggle model browser (1)** puis sur l'icône « Browse masked **subsystems » (2).**



5.5 Lisibilité (cont.)

- **Zoom sur le modèle**
  - Le zoom sur le modèle est accessible à partir du même menu que précédemment : **View/Zoom In** (touche R) et **View/Zoom Out** (touche V)

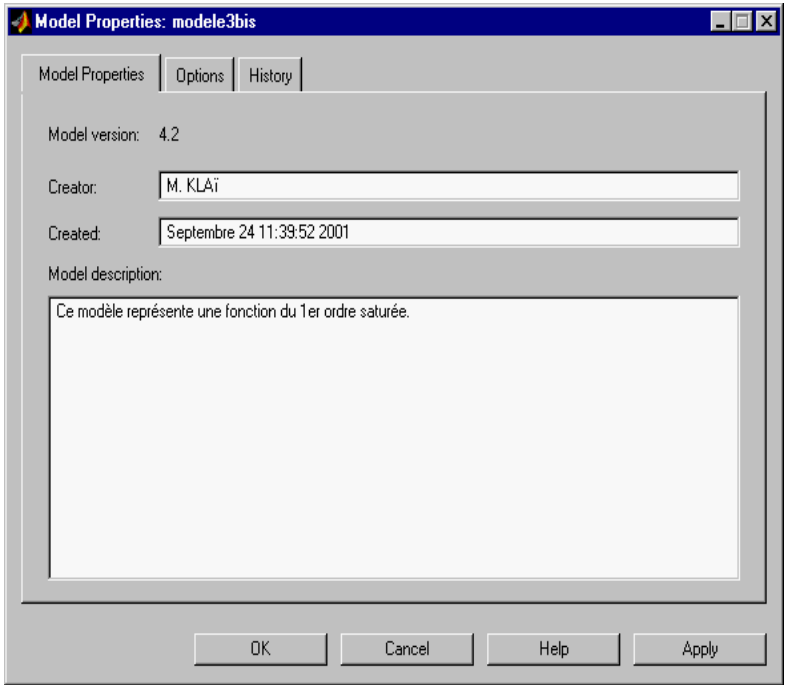


  - Les menus **View/Fit System to View** et **View/Normal (100%)** mettent respectivement le modèle à l'échelle de la fenêtre et à sa taille d'origine.

5.5 Lisibilité (cont.)

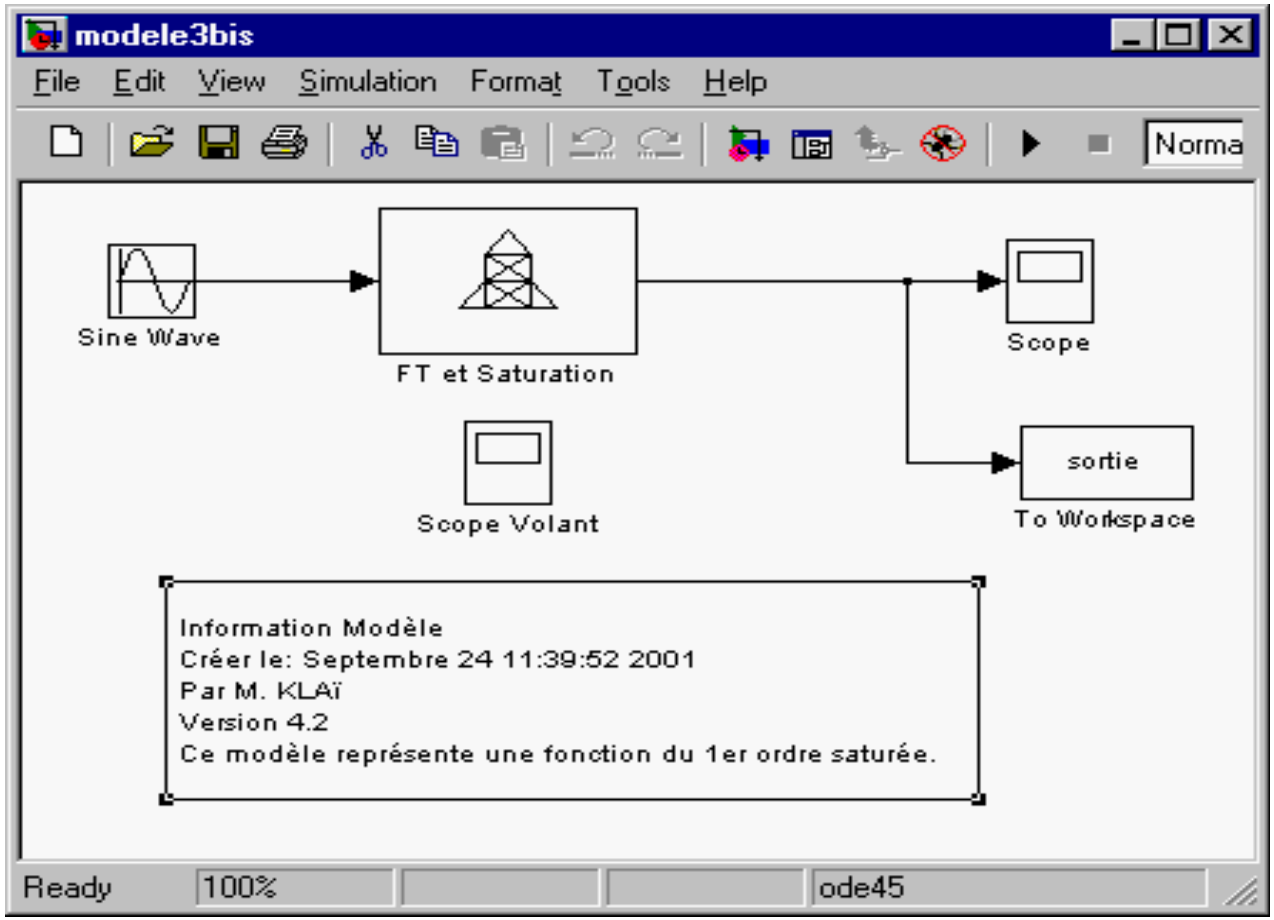
- **Bloc d'Informations**

Le bloc ***Model Info*** regroupe toutes les informations nécessaires à l'identification du modèle : date, auteur, version, mise à jour, etc. Ces différents champs auront été complétés au préalable dans le menu **File/Model Properties**.



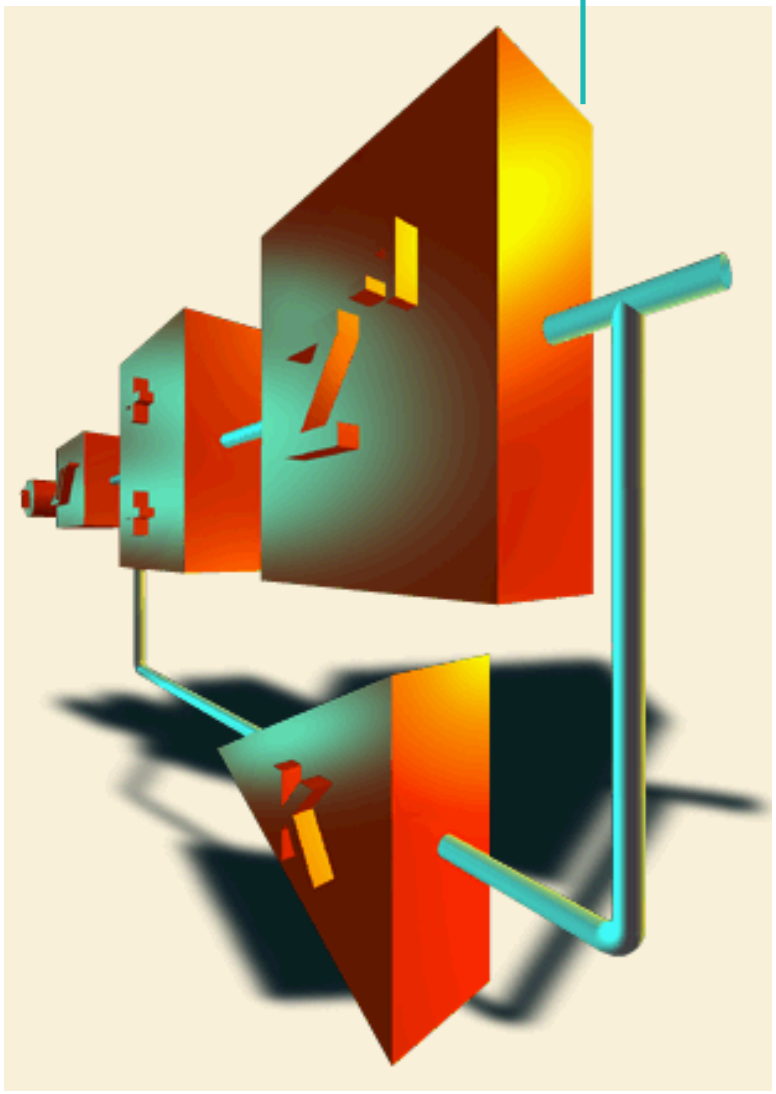
5.5 Lisibilité (cont.)

- **Bloc d'Informations (cont.)**



Simulink

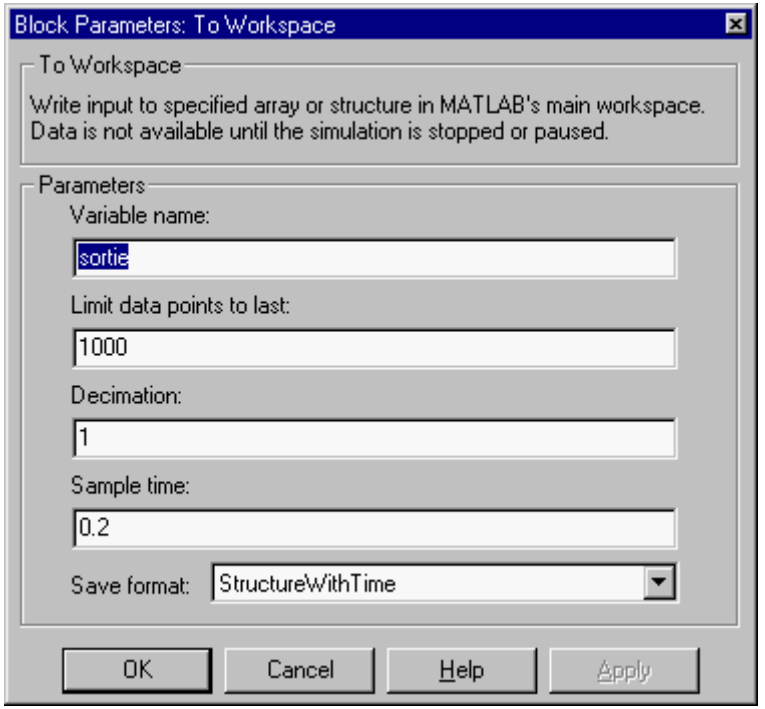
6. Etude à Partir de MATLAB



6.1 Post-traitement dans MATLAB

• Limiter la simulation de *modele1* à 20s.

• Fixer le champ "*Sample time*" du bloc *To Workspace* à 0.2s.



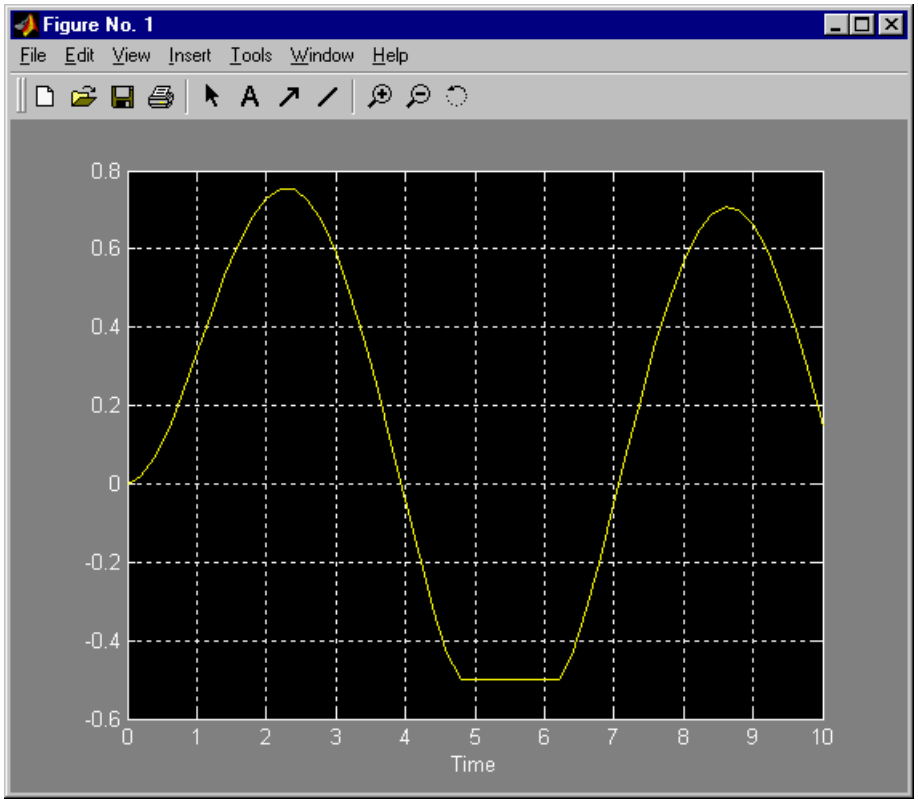
• En arrêtant la simulation, la variable sortie du bloc *To Workspace* est disponible dans MATLAB :

```
>> whos
  Name          Size              Bytes  Class
  sortie        1x1                1476  struct array

Grand total is 127 elements using 1476 bytes
```

6.1 Post-traitement dans MATLAB

• La commande ***simplot*** permet de récupérer les informations de la structure sortie pour tracer le résultat de la simulation dans une fenêtre MATLAB :

```
>> simplot(sortie)
```



6.2 Simulation

• La simulation peut être lancée à partir de MATLAB avec les commandes suivantes :

sim	Simulation
simset	Définition des paramètres de la simulation
simget	Lecture des paramètres de la simulation

Syntaxe de "sim" :

```
>> [t,x,y] = sim('modèle',timespan)
>> [t,x,y] = sim('modèle',timespan,options,ut)
>> [t,x,y1,...,ym] = sim('modèle',timespan,options,ut)
```

avec : **modèle :** nom du modèle

timespan : instant de départ et instant final de la simulation : [TStart TFinal]

options : paramètres de simulation définis par la commande **simset**

ut : tableau de valeurs définissant les entrées externes : **[t,u(1),u(2),...]**

6.2 Simulation (cont.)

Syntaxe de "simset" :

```
>> options = simset
>> options = simset(champ,valeur,...)
>> options = simset(anciennes_options,champ,valeur,...)
```

avec : **champ :** nom du champ

valeur : valeur associée au champ

Syntaxe de "simget" :

```
>> struct_options = simget('modèle',timespan)
>> valeur_options = simget('modèle',champ)
```

avec : **modèle :** nom du modèle

champ : nom du champ désiré

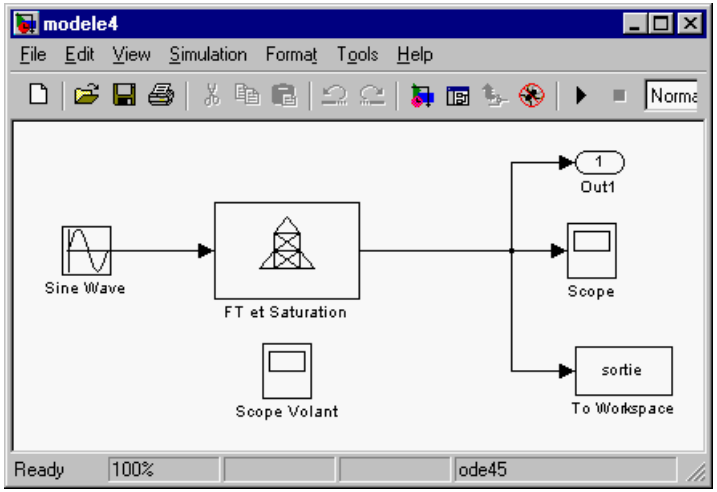
struct_options : structure représentant les options

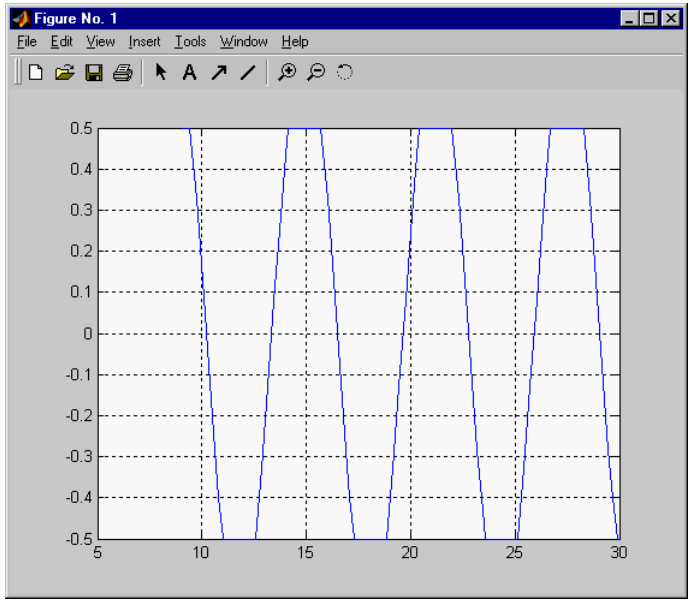
valeur_options : valeur de l 'une des options

6.2 Simulation (cont.)

• Reprendre le modèle précédent en ajoutant un bloc *Outport* pour indiquer la sortie, puis taper :

```
>> options = simget('modele4')
>> mes_options = simset(options,'MaxRows',100,'refine',2)
>> [t,x,y]=sim('modele4',30,mes_options);
>> plot(t,y); grid
```





6.3 Identification des états

vecteur des tailles

Conditions initiales

Noms des états

```
sizes = [
  # états continus
  # états discrets
  # sorties
  # entrées
  # Reservé
  # Couplage entrée/sortie
  # nombre de périodes
    d'échantillonnage
]
```

```
>> [sizes,x0,xord]=modele4
sizes =
     1
     0
     1
     0
     0
     0
     1
x0 =
     0
xord = 'modele4/FT et Saturation/
        Transfer Fcn'
```

Nom du modèle

6.4 Linéarisation

• A partir d'un modèle SIMULINK, il est possible d'extraire une représentation d'état par linéarisation autour d'un point de fonctionnement donné :

$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + DU \end{cases}$$

Syntaxe :

```
>> [A,B,C,D] = linmod('modele')
>> [A,B,C,D] = linmod('modele',x,u)
>> [A,B,C,D] = linmod('modele',x,u,pert)
```

avec `'modèle'` : nom du modèle à linéariser

`x,u` : le point de fonctionnement autour duquel le modèle linéaire doit être extrait est défini par le vecteur d'états x et le vecteur d'entrées u.

`pert` : amplitude de la perturbation appliquée sur les entrées et les états.

Dans le cas de systèmes discrets ou hybrides, utiliser le commande **`dlinmod`**.

6.4 Linéarisation (cont.)

Exemple :

- **L'entrée et la sortie du modèle sont spécifiées par les blocs *Inport* et *Ouport*.**
- **Linéarisation autour de 0 (par défaut) :**

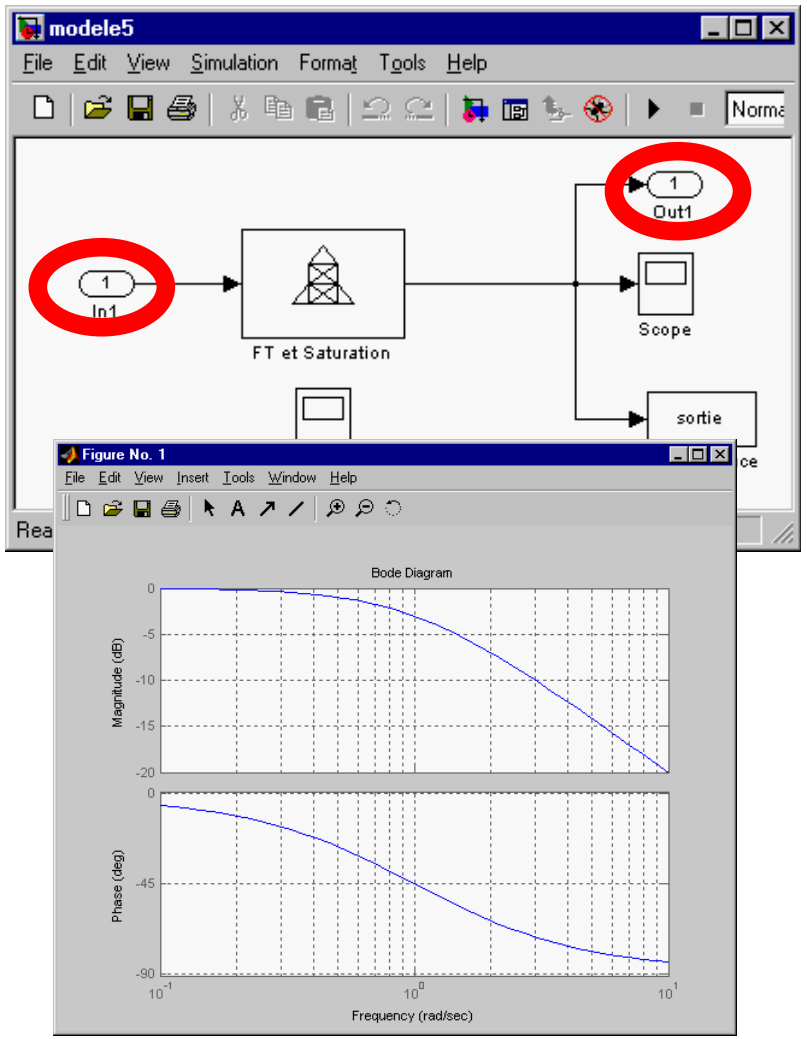
```
>>[A,B,C,D]=linmod('modele5')
```

- **Linéarisation autour de 2 pour l'état et 0 pour l'entrée. La saturation intervient, donc le système n'est plus linéaire : C et D sont nulles.**

```
>>[A,B,C,D]=linmod('modele5',2,0)
```

- **Suite à la linéarisation, il est par exemple possible de tracer le diagramme de Bode :**

```
>> [A,B,C,D]=linmod('modele5');
>> bode(A,B,C,D)
```



6.5 Actions sur un modèle ou un bloc

• Des commandes MATLAB peuvent êtres associées aux blocs et aux modèles. Ces commandes sont exécutées avant ou après certaines actions telles que l'ouverture, la copie ou la sauvegarde du modèle ou du bloc.

Syntaxe :

```
set_param('modèle/bloc','paramètre','action')
```

avec `'modèle/bloc'` : nom du bloc appartenant au modèle.

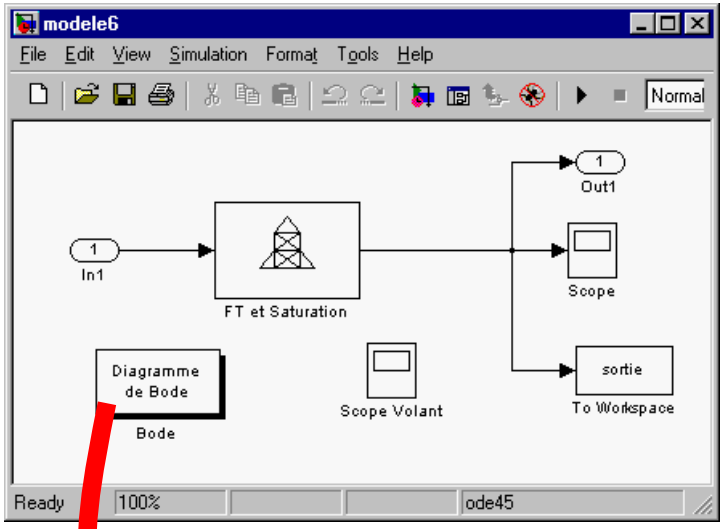
`'paramètre'` : nom du paramètre.

`'action'` : commande MATLAB à associer au paramètre du bloc.

Paramètre	Action à exécuter quand ?
`PreLoadFcn`	avant le chargement du modèle
`PostLoadFcn`	après le chargement du modèle
`CloseFcn`	à la fermeture du modèle ou du bloc
`DeleteFcn`	à la suppression du bloc
`OpenFcn`	à l'ouverture du bloc

6.5 Actions sur un modèle ou un bloc

Exemple : Réalisation d'un bouton de commande pour tracer le diagramme de Bode du modèle.

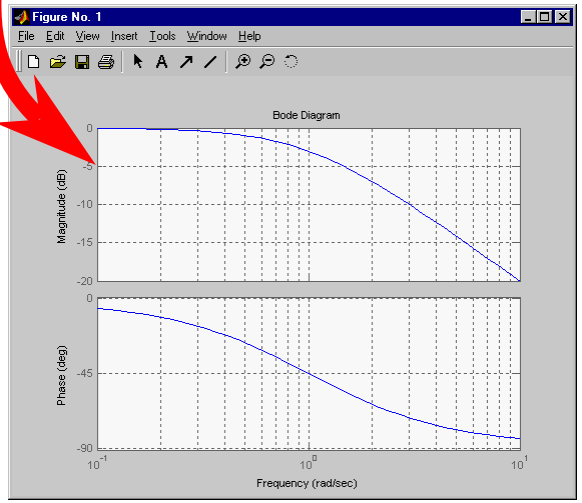


- Reprendre le modèle précédent.
- Ajouter un bloc *Subsystem*.
- Masquer ce bloc en complétant uniquement le champ "*Drawing commands*" par

```
fprintf('Diagramme\nde Bode')
```

- Dans MATLAB, taper la commande suivante :

```
>> set_param('modele6/Bode','OpenFcn',...
'[A,B,C,D]=linmod(''modele6'');bode(A,B,C,D)')
```



6.6 Commandes de construction

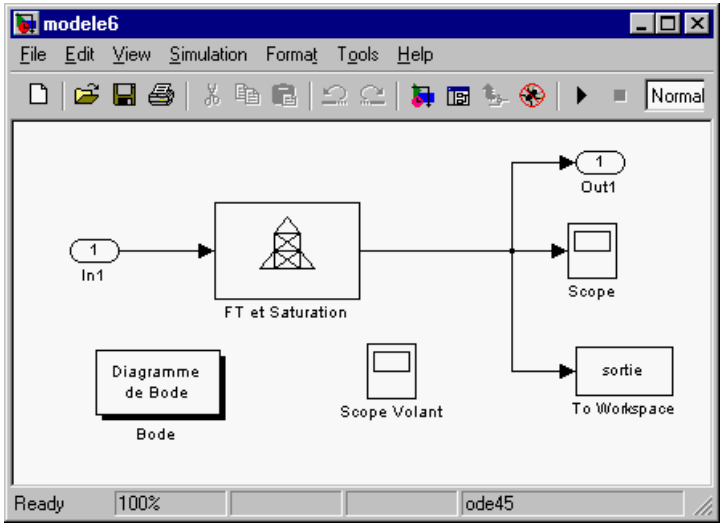
• Des commandes sont utilisées à partir de MATLAB pour modifier un modèle :

open_system	Ouverture d'un système
find_system	Recherche d'un bloc
add_block	Ajout d'un bloc
add_line	Ajout d'une ligne
get_param	Lecture du paramètre d'un bloc
set_param	Ecriture du paramètre d'un bloc
add_param	Ajout de parameters au modèle
delete-param	Efface les paramètres ajoutés par add_param.
gcb	Lecture du nom du bloc courant
gcs	Lecture du nom du système courant
gcbh	Retourne le handle du bloc courant

6.6 Commandes de construction

Exemple :

En reprenant le bouton qui crée le diagramme de Bode, dans le modèle modele6.



La commande associée au bouton est :

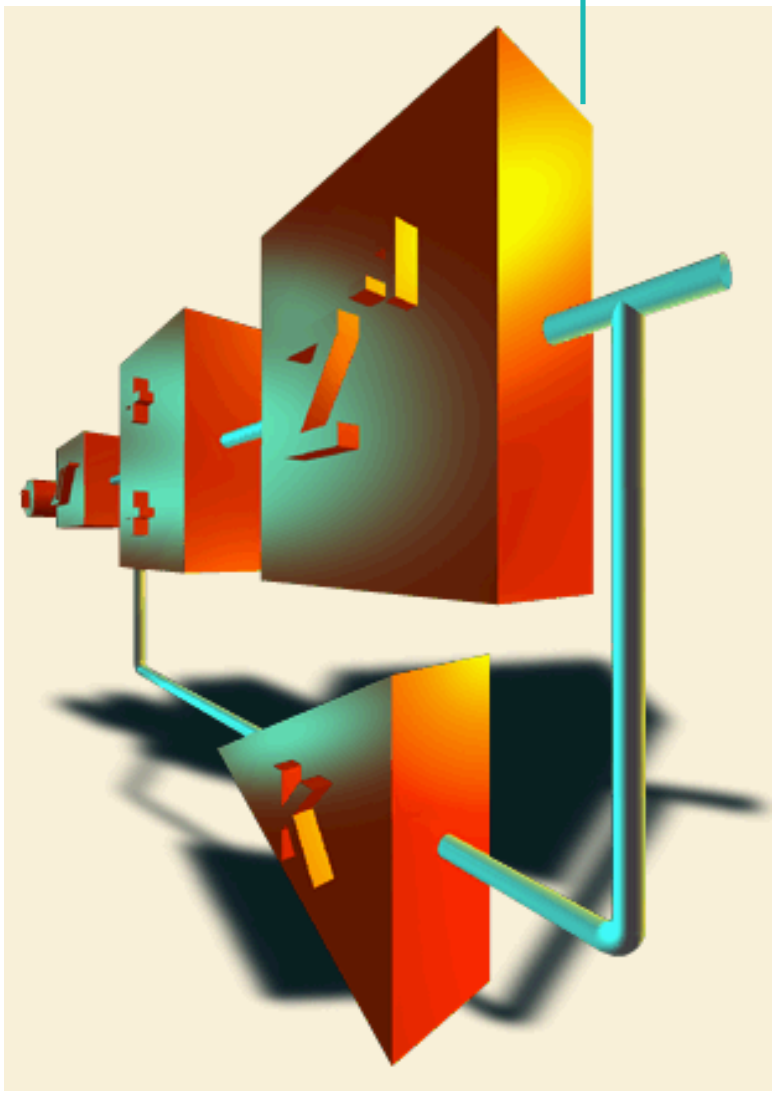
```
>>cmd = get_param('modele6/Bode','OpenFcn')
    cmd =
[A,B,C,D]=linmod('modele6');bode(A,B,C,D)
```

Si le modèle est renommé, la commande associée au bouton devrait être redéfinie. Pour éviter cela, il suffit d'utiliser gcs à la place de 'modele6 ' :

```
>> set_param('modele6/Bode','OpenFcn', . . .
[A,B,C,D]=linmod(gcs);bode(A,B,C,D)')
```

Remarque : dans le cas où l'on dispose de la toolbox ***CONTROL***, SIMULINK donne accès à un champ appelé **Linear Analysis** dans le menu **Tools**. Cet outil est une I.H.M. d'analyse temporelle/fréquentielle permettant une étude complète dans ces domaines en restant dans le même environnement. Il est nécessaire d'insérer dans le modèle des points d'accès pour les entrées/sorties de la linéarisation. Ceci nous permet, entre autre, de réaliser un diagramme de Bode directement à partir du modèle.

7. Moteur à Courant Continu



7.1 Equations Physiques

En mécanique, le couple en sortie du moteur s'écrit sous la forme suivante :

$$\Gamma(t) = J\frac{d\omega(t)}{dt} + f\cdot\omega(t)$$

Γ : couple en N.m
J : inertie en kg.m^2
ω : vitesse angulaire en rad.s^{-1}
f : coef. de frottement visqueux en N.m.s

En électricité, l'expression du couple est la suivante :

$$\Gamma(t) = Kc\cdot I(t)$$

Kc : constante de couple en N.m.A^{-1}
I : courant d 'induit en A

La tension en entrée du moteur s'écrit :

$$U(t) = r\cdot I(t) + L\cdot\frac{dI(t)}{dt} + Ke\cdot\omega(t)$$

r : résistance d 'induit en Ω
L : inductance d 'induit en H
Ke : réaction d 'induit en N.m.A^{-1}

Données numériques :

J : 0.1 kg.m^2 Kc : 0.5 N.m.A^{-1} r : 0.1 Ω
f : 0.01 N.m.s Ke : 0.1 N.m.A^{-1} L : 0.5 H

7.2 Modélisation

- **Représentation d'état du système**

Les équations peuvent être mises sous la forme suivante :

$$\begin{cases} J \cdot \dot{\omega}(t) + f \cdot \omega(t) = Kc \cdot I(t) \\ r \cdot I(t) + L \cdot \dot{I}(t) = U(t) - Ke \cdot \omega(t) \end{cases}$$

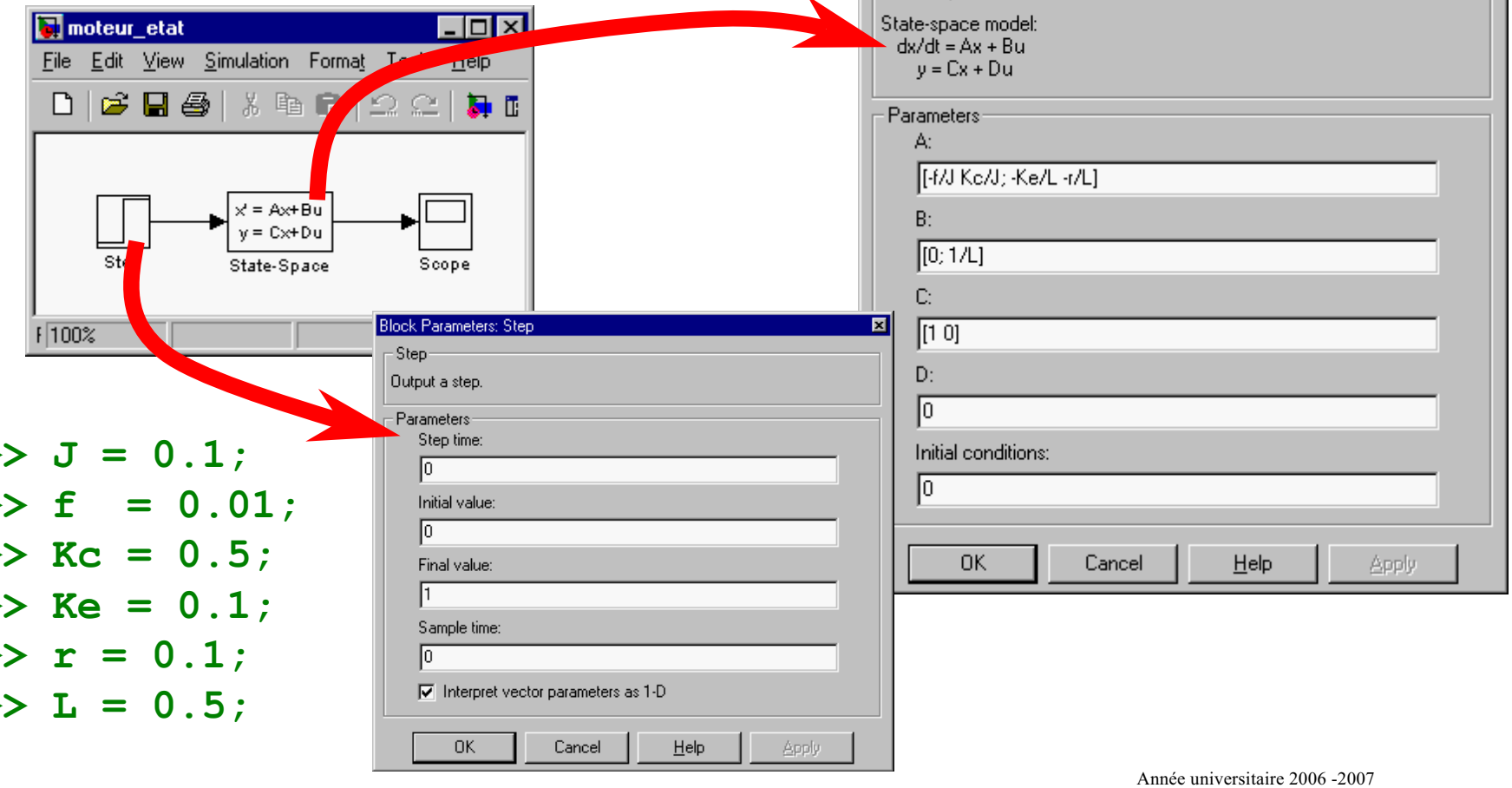
$$\begin{cases} \dot{\omega}(t) = -\frac{f}{J} \cdot \omega(t) + \frac{Kc}{J} \cdot I(t) \\ \dot{I}(t) = -\frac{Ke}{L} \cdot \omega(t) - \frac{r}{L} \cdot I(t) + \frac{U(t)}{L} \end{cases}$$

$$\begin{cases} \frac{d}{dt}\begin{bmatrix} \omega(t) \\ I(t) \end{bmatrix} = \begin{bmatrix} -\frac{f}{J} & \frac{Kc}{J} \\ -\frac{Ke}{L} & -\frac{r}{L} \end{bmatrix} \cdot \begin{bmatrix} \omega(t) \\ I(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} \cdot U(t) \\ \omega(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \omega(t) \\ I(t) \end{bmatrix} + 0 \cdot U(t) \end{cases}$$

7.2 Modélisation (cont.)

- **Représentation d'état du système (cont.)**

Le modèle suivant est créé :

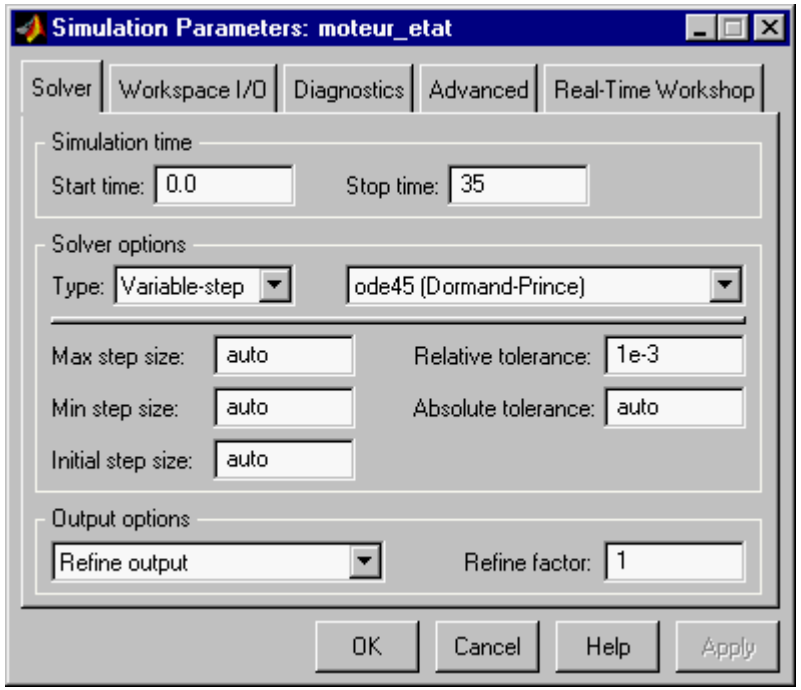


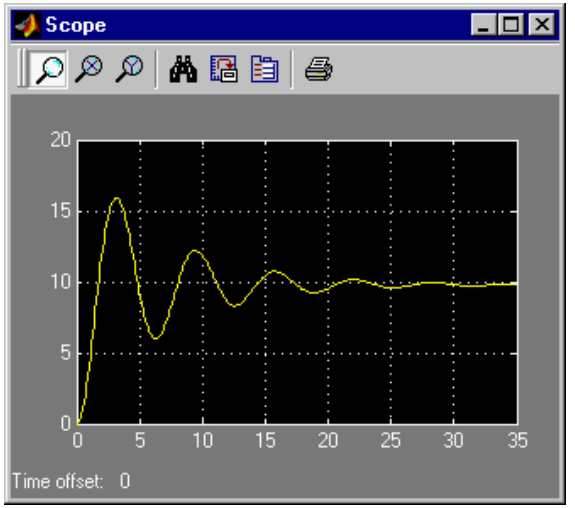

```
>> J = 0.1;
>> f  = 0.01;
>> Kc = 0.5;
>> Ke = 0.1;
>> r = 0.1;
>> L = 0.5;
```

7.2 Modélisation (cont.)

- **Représentation d'état du système (cont.)**

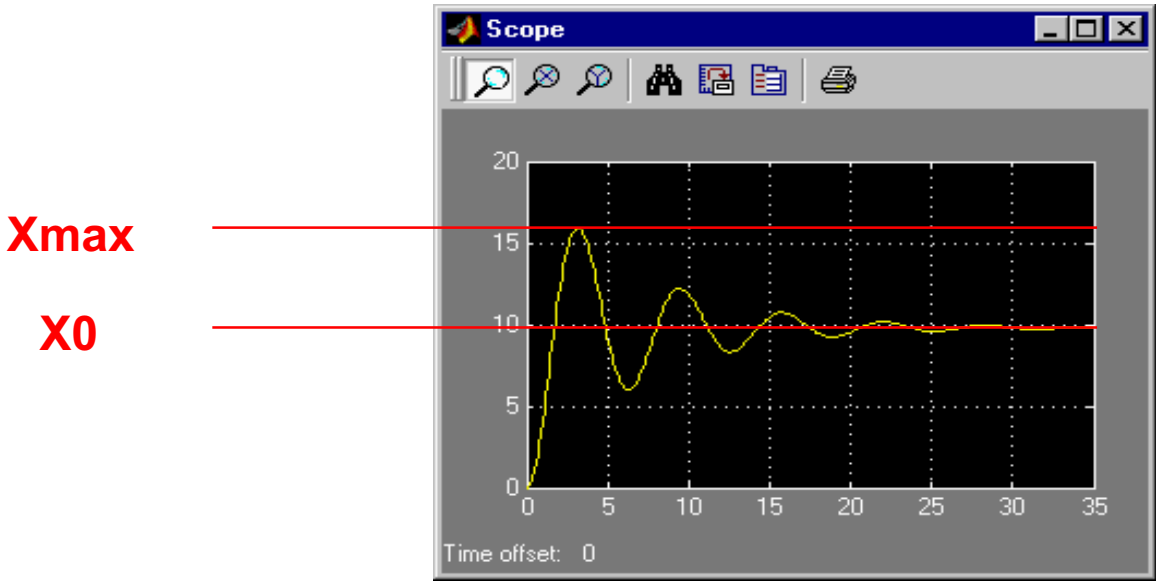
Les paramètres de simulation sont les suivants :





7.2 Modélisation (cont.)

- **Représentation d'état du système (cont.)**



<u>Exercice</u> :

- **Mesurer le dépassement de la réponse indicielle du moteur : d=(Xmax-X0)/X0**
- **Mesurer le temps de réponse à 5%**

7.2 Modélisation (cont.)

- **Représentation sous forme de schémas-blocs**

Les équations du moteur sont :

$$J\frac{d\omega(t)}{dt} + f\cdot\omega(t) = Kc\cdot I(t)$$

$$r\cdot I(t) + L\cdot\frac{dI(t)}{dt} = U(t) - Ke\cdot\omega(t)$$

La transformée de Laplace de ces équations, avec $\omega(t=0+)=0$ et $I(t=0+)=0$, s'écrit :

$$J\cdot p\cdot\omega(p) + f\cdot\omega(p) = Kc\cdot I(p)$$

$$r\cdot I(p) + L\cdot p\cdot I(p) = U(p) - Ke\cdot\omega(p)$$

Soit $U(t)$ l'entrée et $\omega(t)$ la sortie de notre modèle, les expressions précédentes se mettent sous la forme suivante :

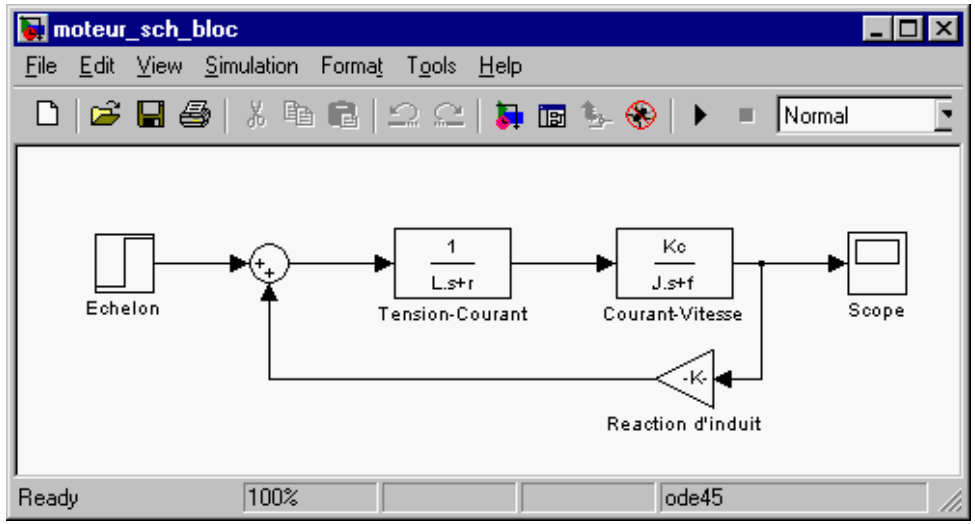
$$\omega(p) = \frac{Kc}{J\cdot p + f}\cdot I(p)$$

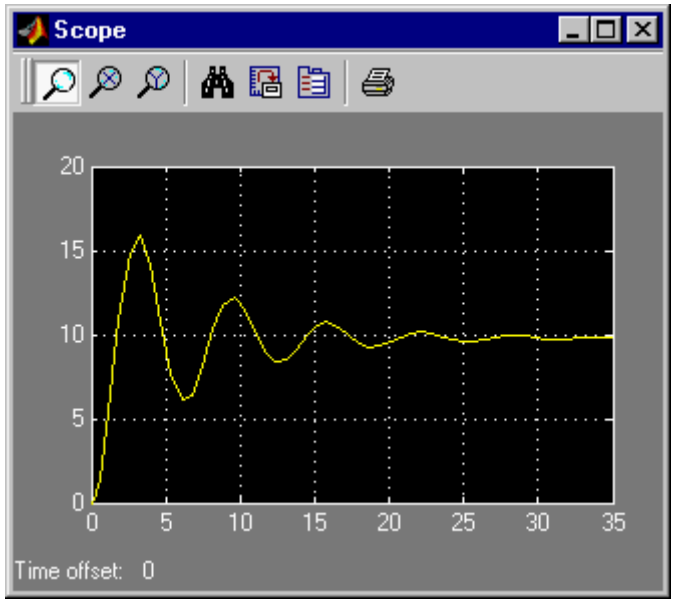
$$I(p) = \frac{1}{L\cdot p + r}\left[U(p) - Ke\cdot\omega(p)\right]$$

7.2 Modélisation (cont.)

- **Représentation sous forme de schémas-blocs**

A partir des expressions précédentes, le modèle suivant est créé :





En conservant les paramètres définis précédemment, la réponse indicielle du moteur est la même.

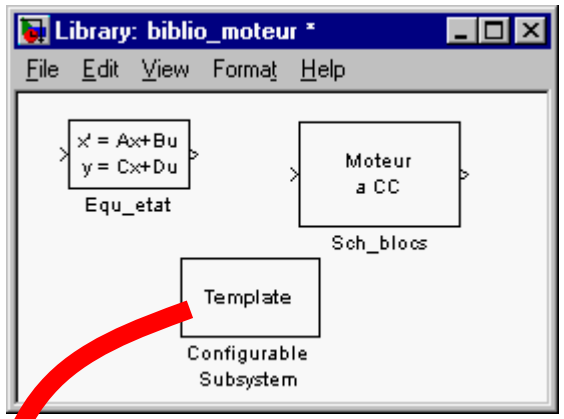
7.2 Modélisation (cont.)

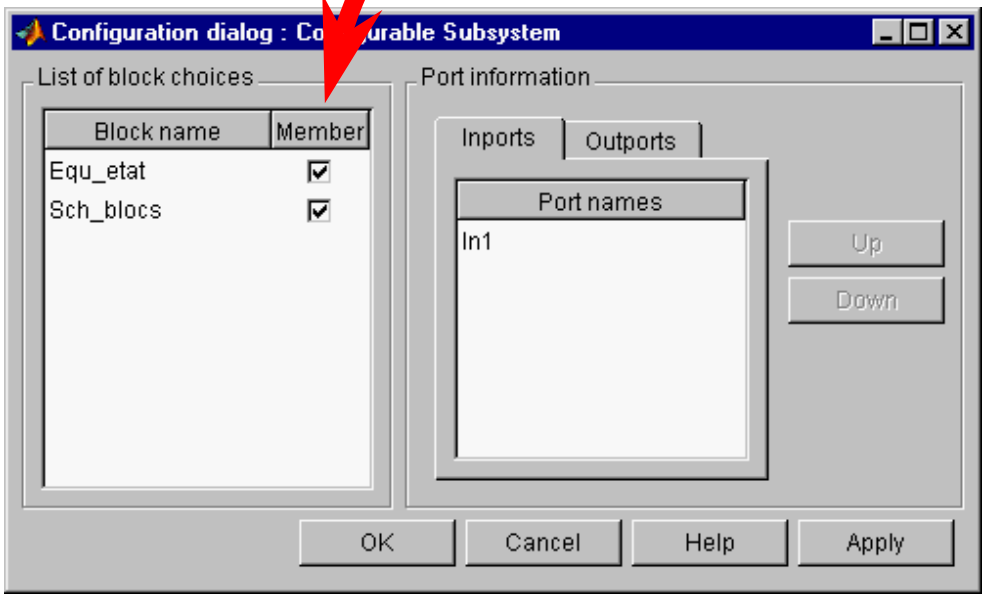
- **Validation des modèles**

Après avoir créé différents modèles, nous allons maintenant les valider par l'intermédiaire du bloc *Configurable Subsystem*.

Il faut d'abord créer une bibliothèque pour rassembler nos modèles :

- Sélectionner **File\New\Library**
- Déposer, dans cette bibliothèque, les modèles du moteur à courant continu
- Déposer le bloc ***Configurable Subsystem*** de la bibliothèque ***Signals&Systems*** dans la bibliothèque en cours de construction
- Enregistrer le tout sous ***biblio_moteur***

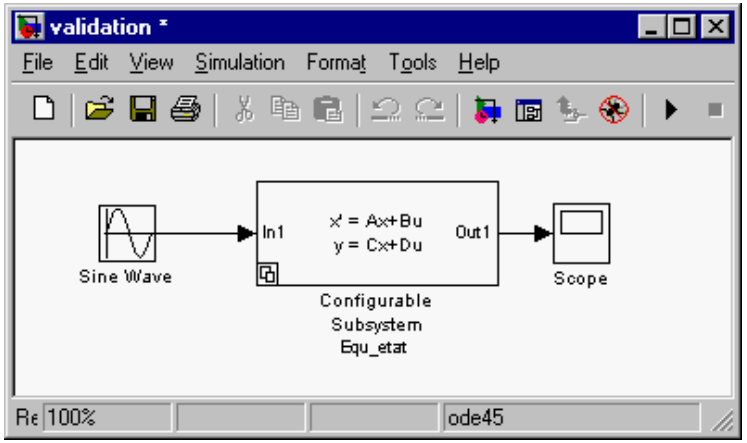


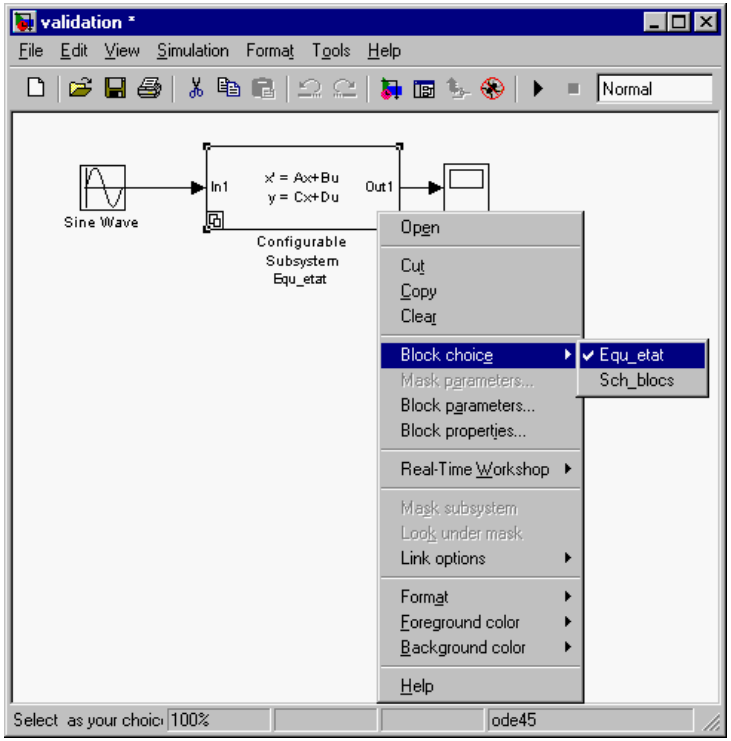


7.2 Modélisation (cont.)

- **Validation des modèles (cont.)**

Il nous reste à créer le modèle avec les 3 blocs *Sine Wave*, *Configurable Subsystem* et *Scope*, puis configurer le sous-système comme ci-contre.





Nous disposons, maintenant, d'un système qui nous permet de choisir un bloc dans la bibliothèque et de le configurer sans déplacement de blocs.

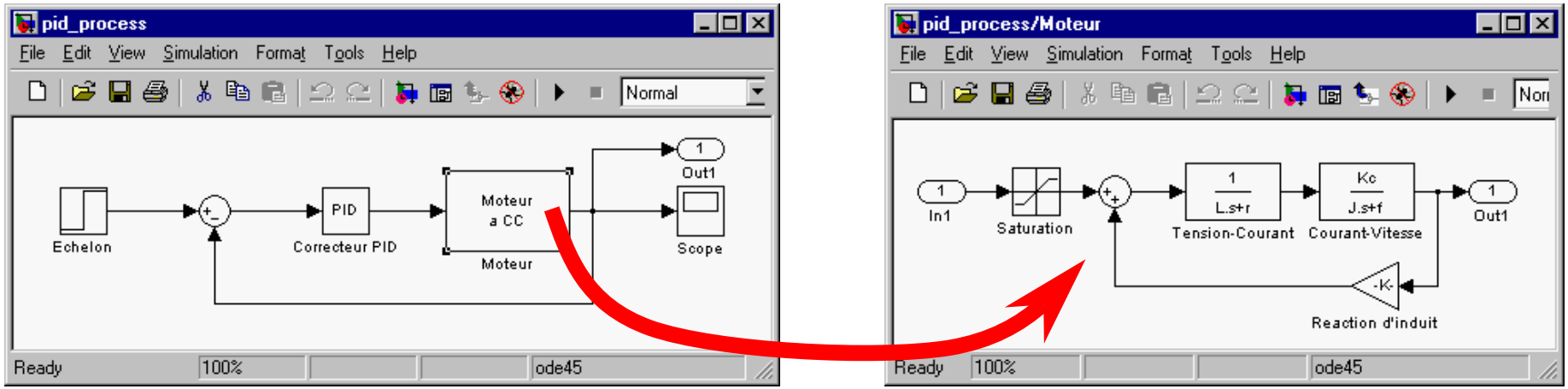
7.3 Régulation par PID

- **Le problème**

La condition suivante est à prendre en compte : la tension à l'entrée du moteur ne peut pas dépasser la valeur de 0.5 V (elle sature).

La réponse indicielle oscille trop, c'est pourquoi un contrôleur PID est utilisé pour stabiliser le procédé. Quelles doivent être les valeurs des trois gains du PID vérifiant le critère de la minimisation de l'erreur quadratique moyenne ?

Considérons la boucle de régulation suivante :

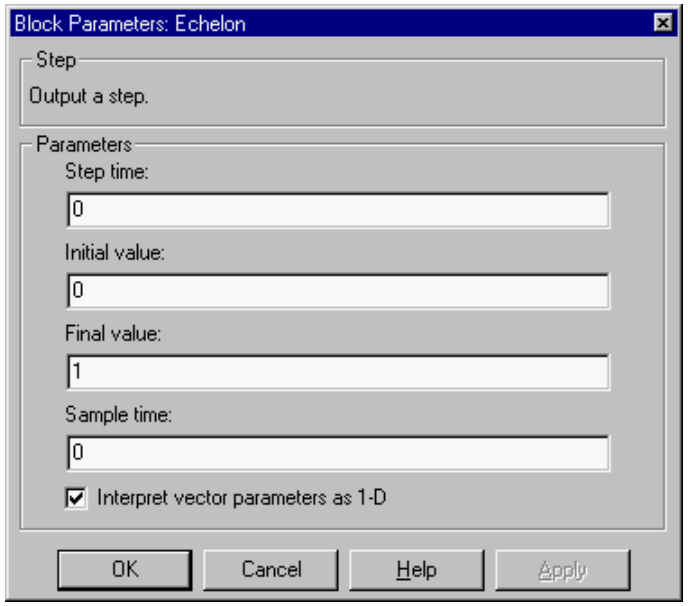


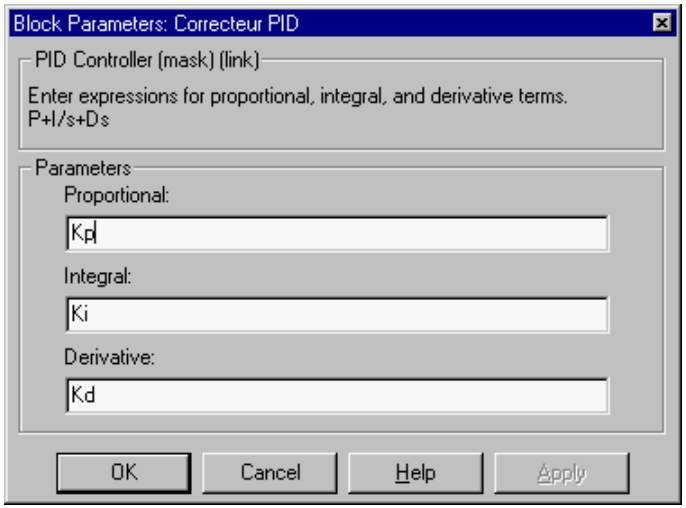
La saturation insérée en entrée du modèle ne permet pas d'utiliser les méthodes issues de la théorie linéaire pour le calcul des coefficients du PID (méthode de Ziegler-Nichols par exemple). Nous avons donc recours à une méthode d 'optimisation numérique.

7.3 Régulation par PID (cont.)

- **Le problème (cont.)**

L'échelon passe de l'amplitude 0 à l'amplitude 1 à T=0s





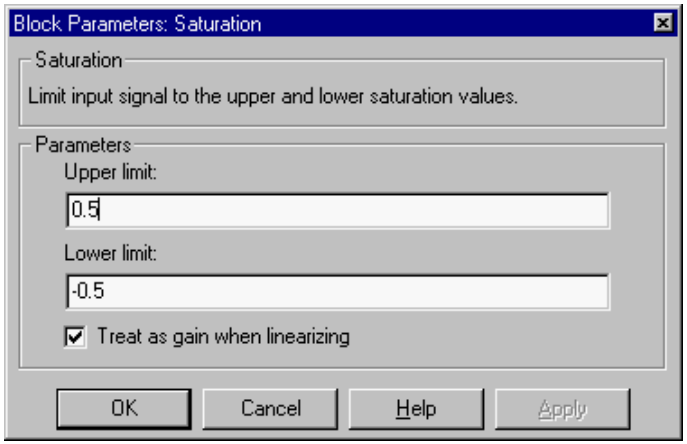
Le régulateur PID vient de la bibliothèque *Simulink_Extras/ Additional_Linear*. C'est un bloc masqué constitué, d'un gain, d'un intégrateur et d'un dérivateur. Il est paramétré avec les trois variables :

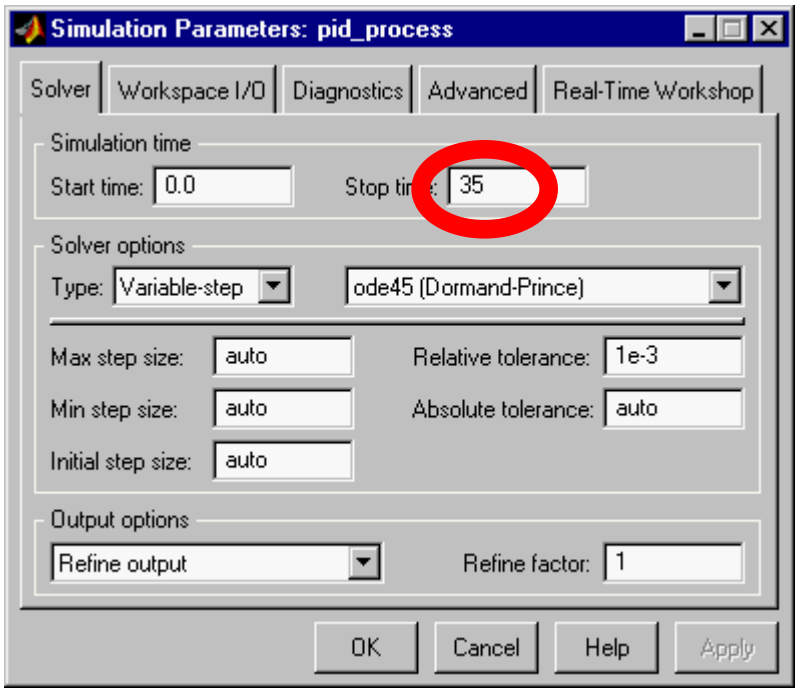
- **Kp : gain du bloc proportionnel**
- **Ki : gain du bloc intégrateur**
- **Kd : gain du bloc dérivateur**

7.3 Régulation par PID (cont.)

- **Le problème (cont.)**

Le bloc *Saturation* limite sa sortie entre -0.5 et 0.5





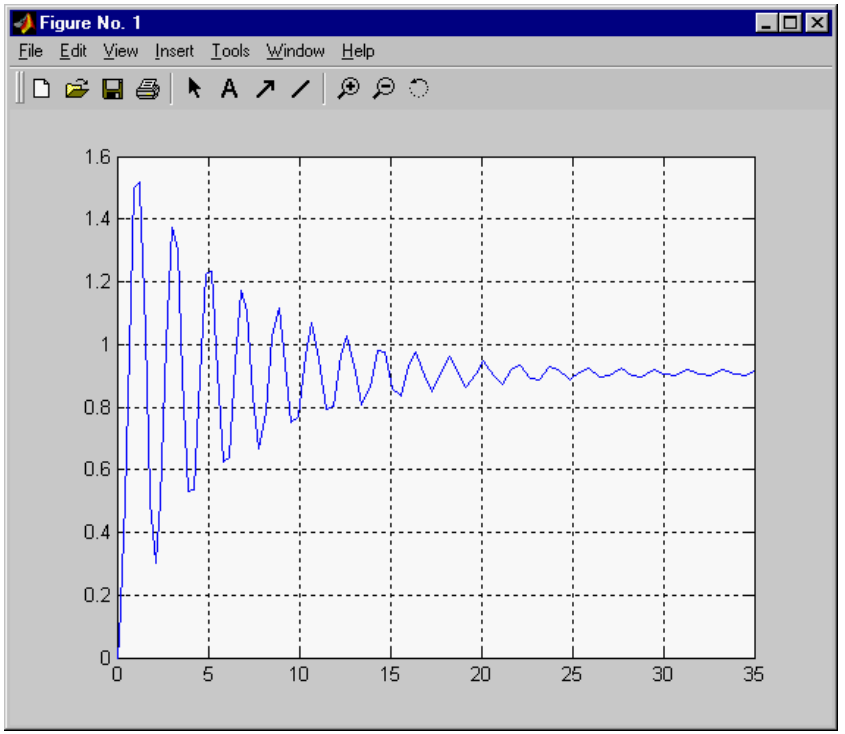
<u>Exercice</u> : dans le bloc *PID Controller*, remplacez les trois paramètres par des valeurs de manière à stabiliser le système. Par exemple, vous pouvez commencer par remplacer Kp, Ki et Kd par 1, 0 et 0.

7.3 Régulation par PID (cont.)

- **Etude paramétrique**

Exemple 1 : Simulation avec Kp, Ki et Kd fixés à 1, 0 et 0.

```
>> Kp=1;
>> Ki=0;
>> Kd=0;
>> [t,x,y]=sim(' pid_process ',35);
>> plot(t,y);grid
```

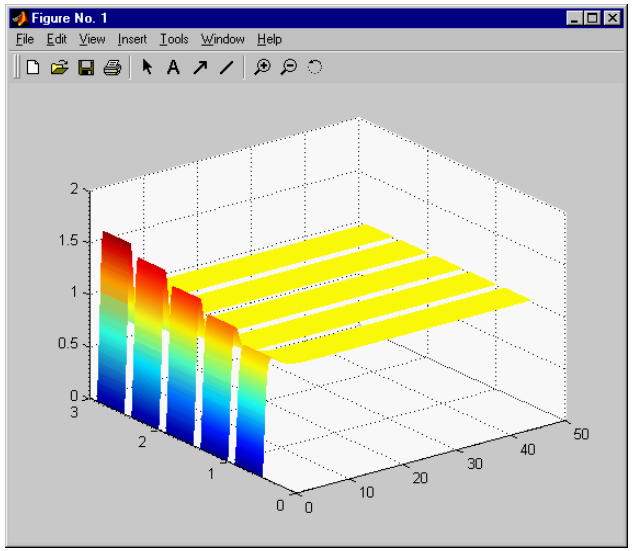


7.3 Régulation par PID (cont.)

- **Etude paramétrique (cont.)**

Exemple 2 : Etude paramétrique en faisant varier le gain du bloc intégrateur :

```
>> etud_param
```



D:\Formation\etud_param.m*

File Edit View Text Debug Breakpoints Web Window Help

```
%%%%%%%%%%%%%%%%%%%%%%%
% Programme etud_param %
%%%%%%%%%%%%%%%%%%%%%%%
% Augmentation du gain intégral et présentation
% du résultat sur des bandes juxtaposées

% Superposition de courbes
figure;
hold on;

% Gains fixes
Kp=1;
Kd=0.5;

% Boucle de simulations:
% Variation sur Ki=0.5..2.5
for i=1:5,
   Ki=i/2;
   % Simulation
   [t,x,y]=sim('pid_process',50);
   % Affichage du résultat sous forme d'une
   % bande
   surf([t t], ones(size(y))*[Ki Ki+0.4], [y y])
end

view(3);
grid;
shading interp;
```

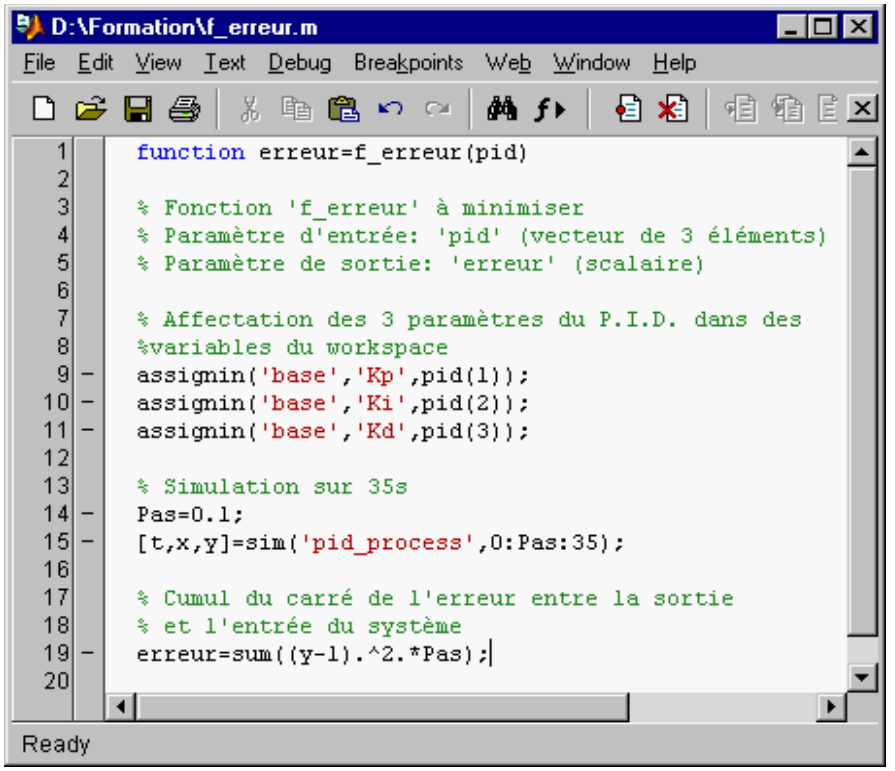
Ready

7.3 Régulation par PID (cont.)

- **Optimisation numérique de la régulation**

MATLAB va être utilisé pour déterminer les trois gains du PID, Kp Ki Kd, qui minimisent l'erreur de position, c'est-à-dire l'écart entre l'entrée du système (échelon) et la sortie du système (réponse à l'échelon).

L'erreur de position est calculée par la fonction suivante :



Exemple :

```
>> f_erreur([1 0 0])
ans =
      1.5585

>> f_erreur([1 0 1])
ans =
      0.9609
```

7.3 Régulation par PID (cont.)

- **Optimisation numérique de la régulation (cont.)**

L'optimisation du PID se fait par la fonction **fminsearch** : minimisation d'une fonction multivariable.

Le programme d'optimisation est le suivant :

D:\Formation\optimise_pid.m*

File Edit View Text Debug Breakpoints Web Window Help

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Programme d'optimisation du P.I.D. %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Valeurs initiales de Kp, Ki et Kd
pid_init=[1 1 1];

% Criteres pour l'optimisation:
%     pas d'affichage
%     critere de tolerance sur l'entree de la fonction
%     critere de tolerance sur la sortie de la fonction
options=optimset('Display','final','TolX',0.001,'Tolfun',0.001);

% Processus d'optimisation
pid=fminsearch('f_erreur',pid_init,options)
```

Ready

7.3 Régulation par PID (cont.)

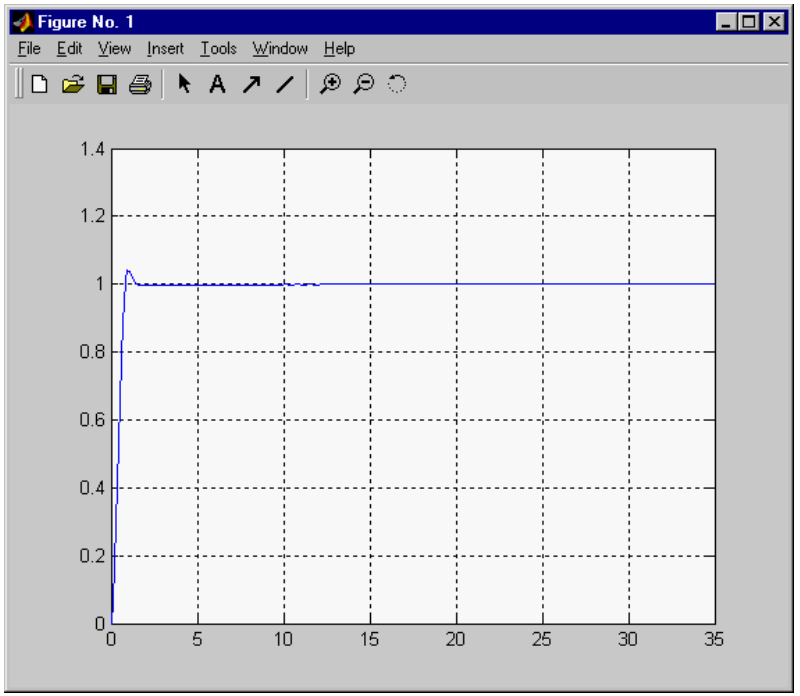
- **Optimisation numérique de la régulation (cont.)**

En lançant **optimise_pid**, les résultats obtenus sont :

2.8055 0.2177 0.6120

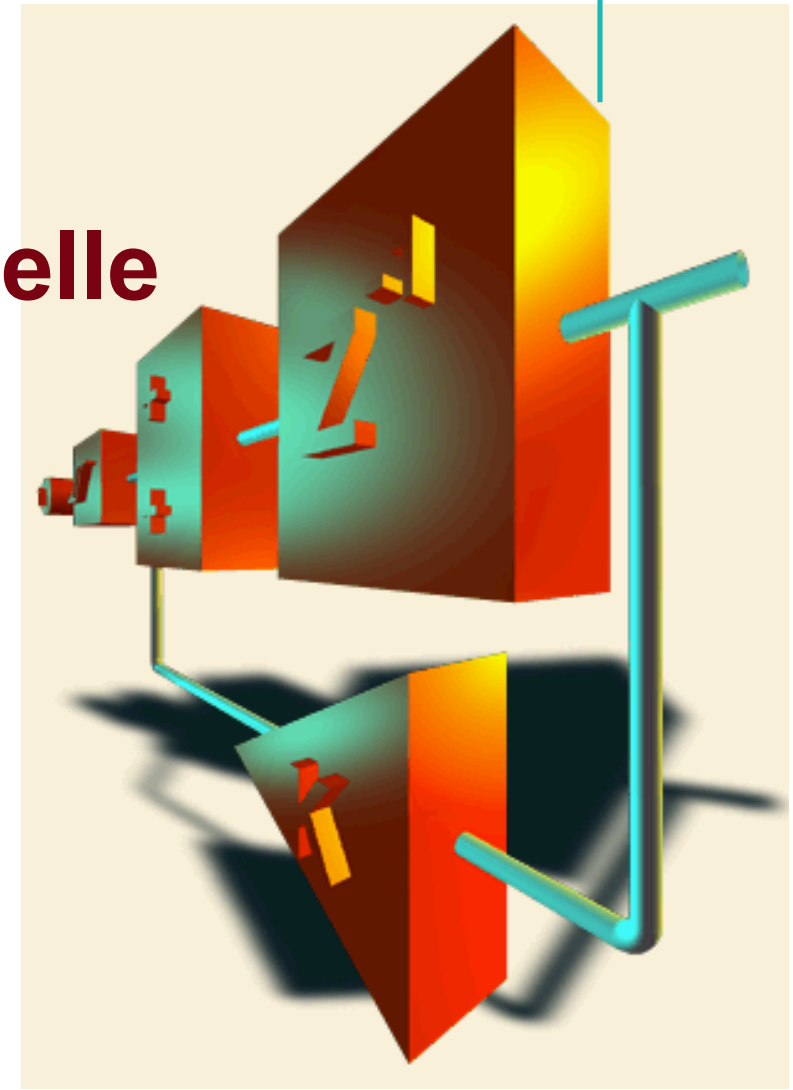
La réponse indicielle est alors :

```
>> Kp=pid(1);
>> Ki=pid(2);
>> Kd=pid(3);
>> [t,x,y] = sim('pid_process',35);
>> plot(t,y);grid
```



Remarque : **fminsearch** utilise un algorithme d'optimisation sans contrainte pouvant conduire à des valeurs de gains négatives. Pour prendre en compte les contraintes il faut utiliser **fmincon** de la **optimization toolbox**.

8. Sous-systèmes à exécution conditionnelle



8.1 Présentation

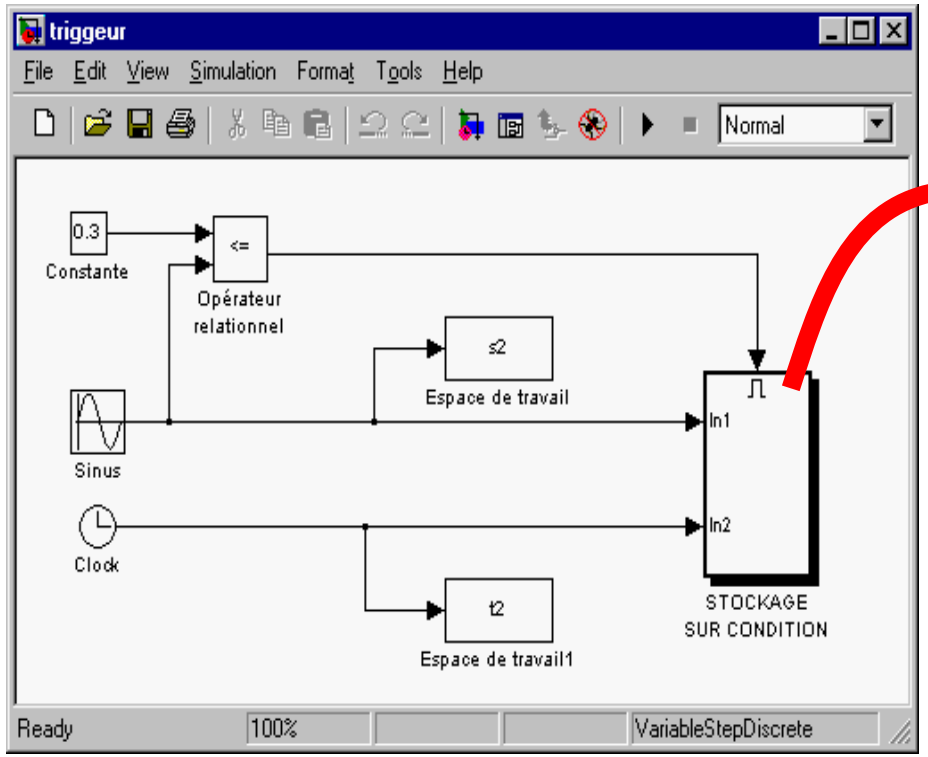
Un sous-système *à exécution conditionnelle* permet de simuler des modèles SIMULINK sur événement :

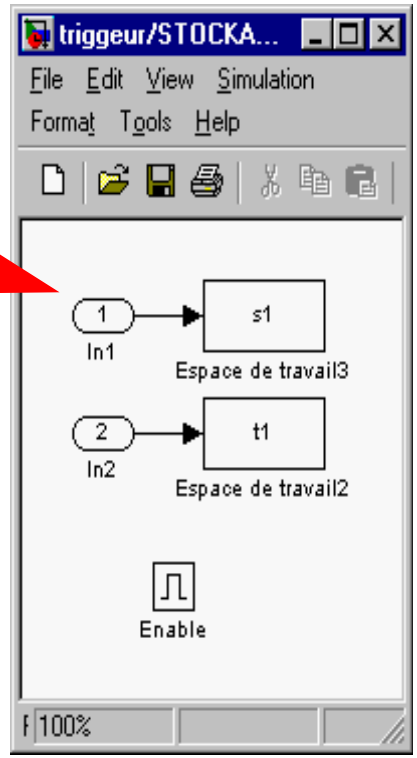
- **Sur niveau : bloc Enable**
- **Sur front montant ou descendant : bloc « Trigger »**
- **Sur test : blocs IF/ELSE, bloc SWITCH CASE**

8.2 Exemple 1 : stockage conditionnel

- **Les valeurs sont enregistrées uniquement lorsque le signal sortant du générateur de signaux est supérieur à 0.3.**

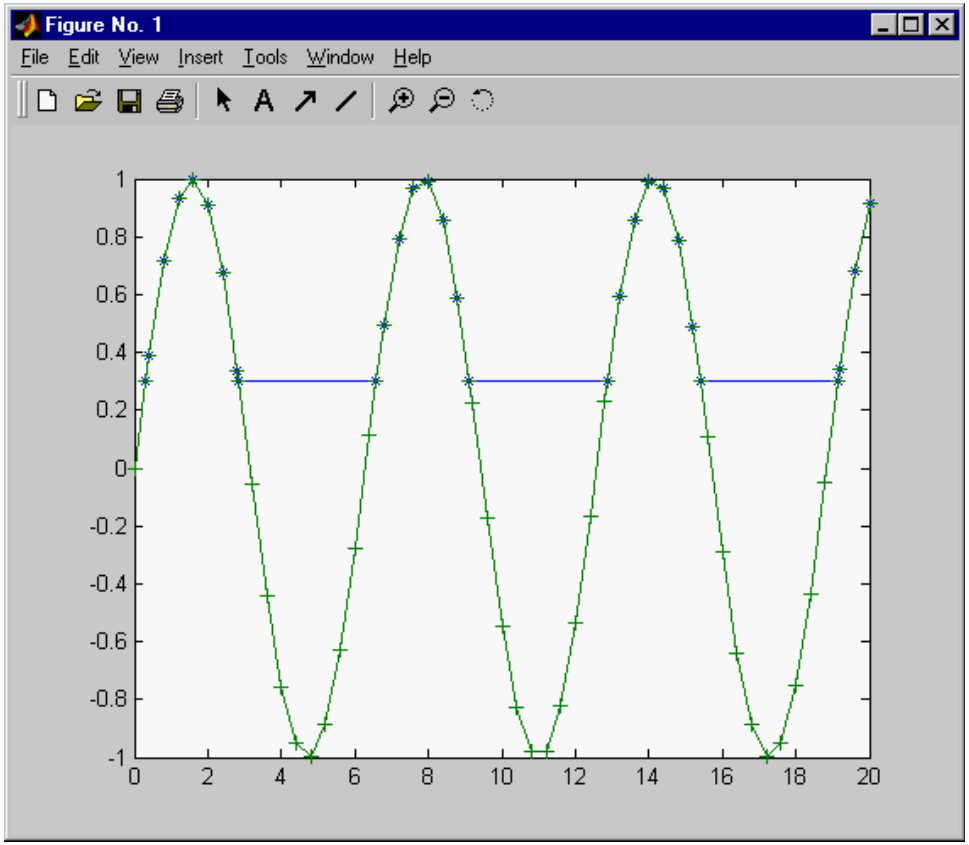
  - **Utilisation du Enabled subsystem**





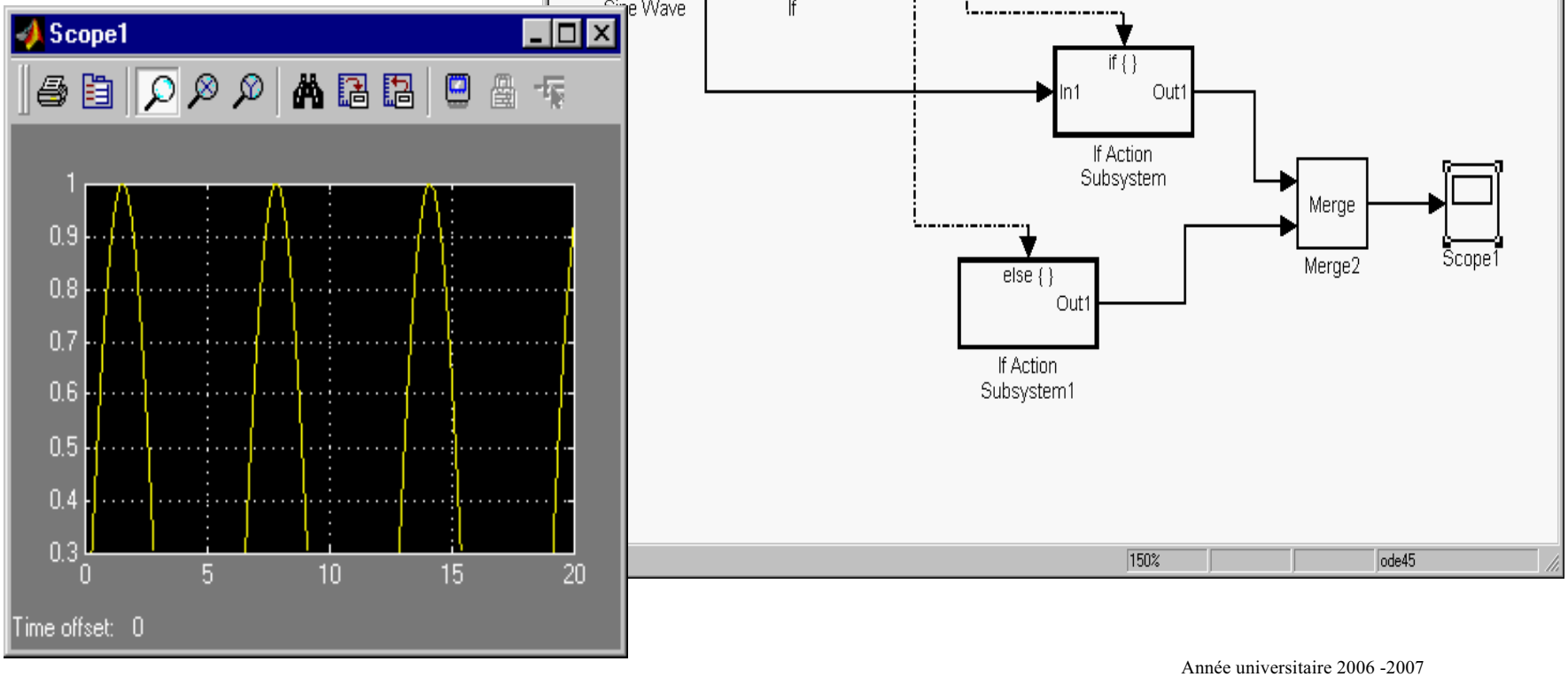
8.2 Exemple 1 : stockage conditionnel

```
>>plot(t1,s1,'x-',t2,s2,'+-')
```



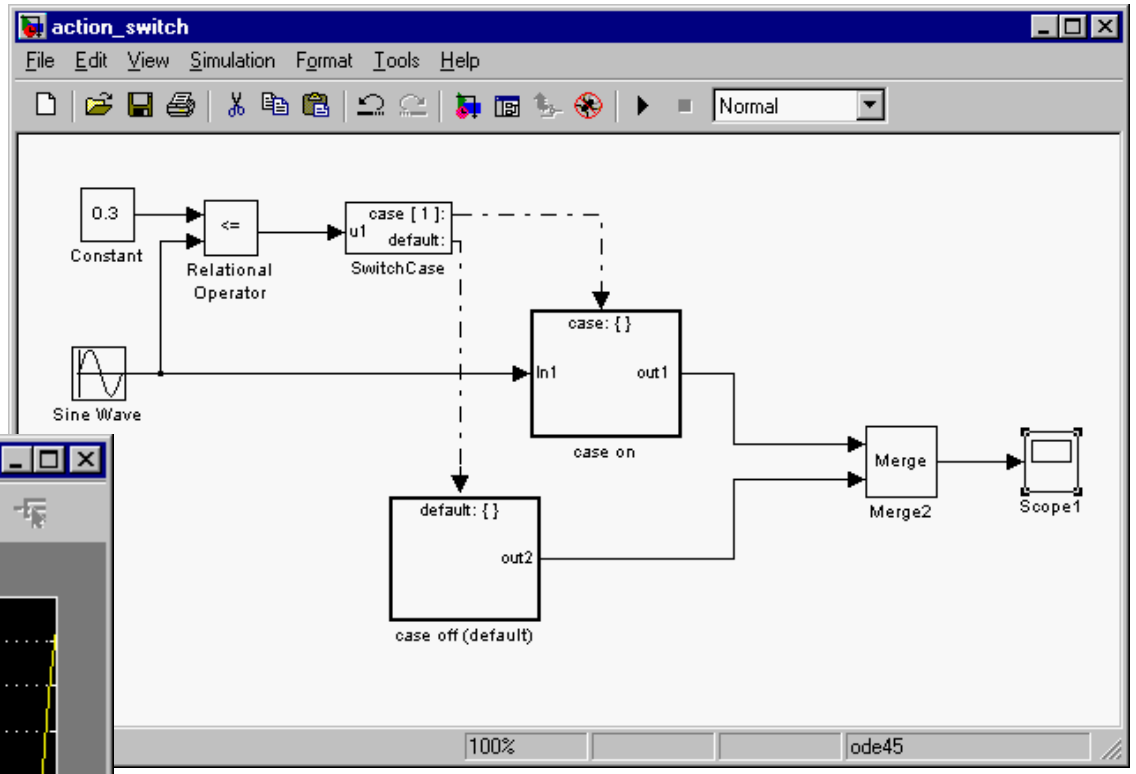
8.2 Exemple 1: stockage conditionnel

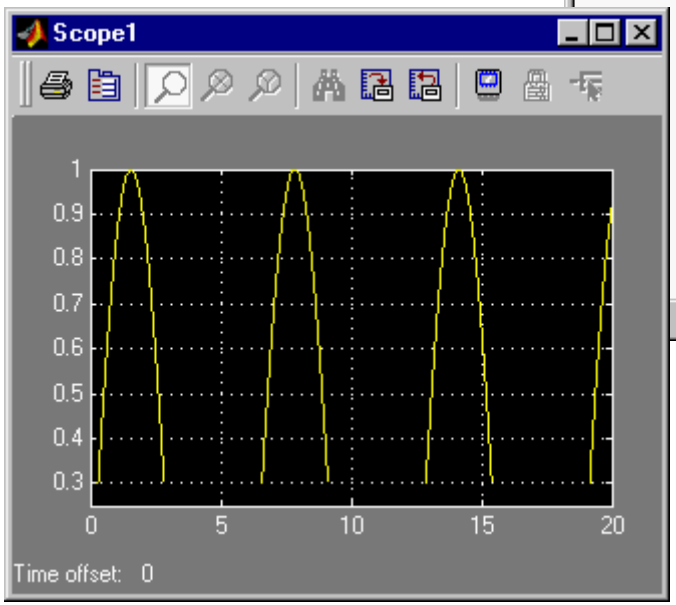
- **Utilisation du IF/ELSE**



8.2 Exemple 1: stockage conditionnel

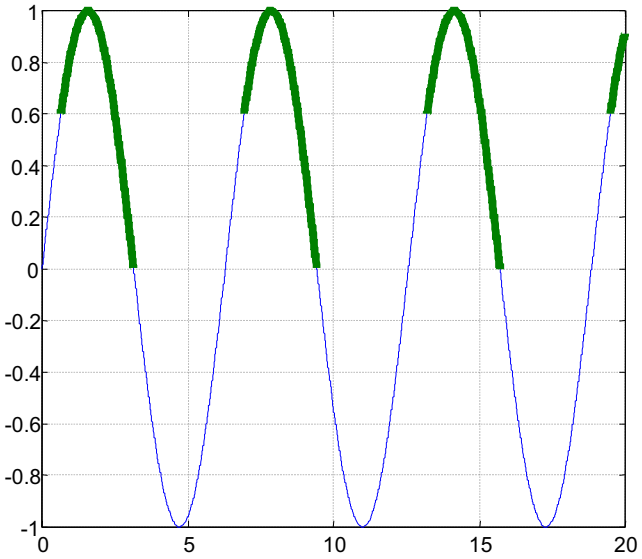
- **Utilisation du SWITCH/CASE**





Exercice 1

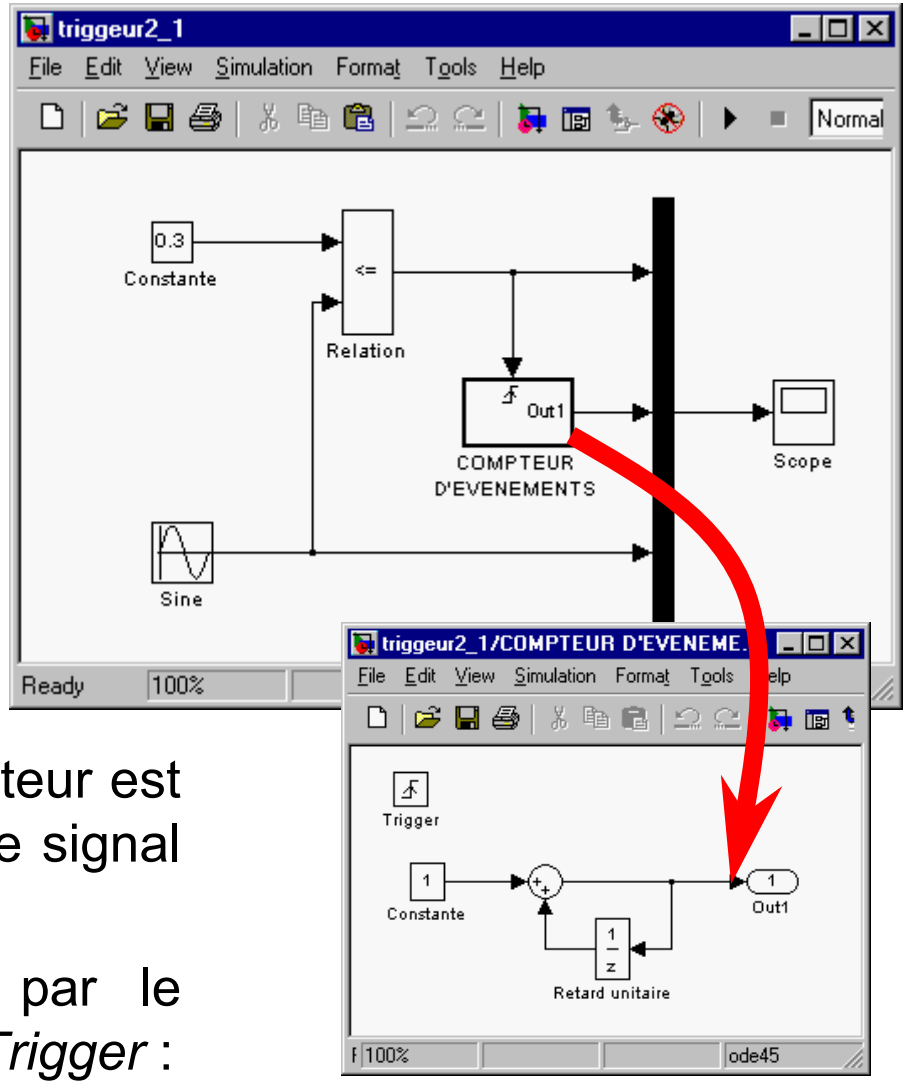
- **Modifier le modèle pour que le stockage des valeurs soit déclenché lorsque le signal passe au-dessus de 0.6 et arrêté lorsque le signal passe au-dessous de la valeur 0.**
- **Comment obtenir cet affichage (voir graphique ci-dessous) : le trait épais représente la sortie du système "enabled".**



8.3 Exemple 2 : Compteur d'évènements

• Les blocs triggés peuvent être utilisés pour la réalisation d'un compteur d'événements sur fronts montants et/ou descendants.

Le schéma utilisant le compteur est le suivant :

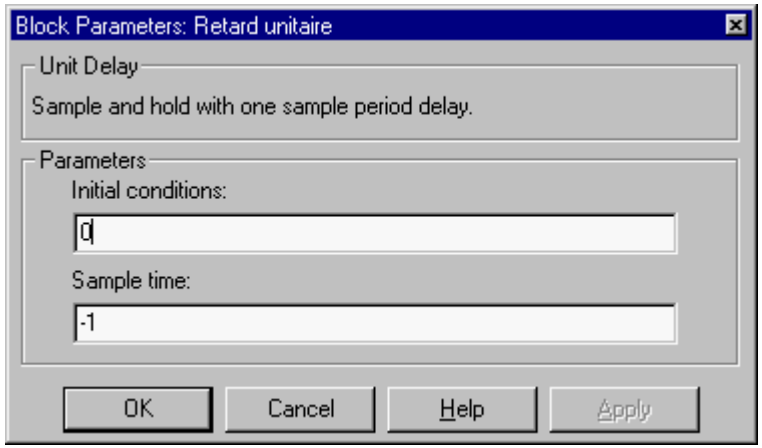


• Le signal déclenchant le compteur est celui issu du bloc relationnel : le signal est comparé à la valeur 0.3.

Le compteur est représenté par le *Subsystem* qui contient le bloc *Trigger* :

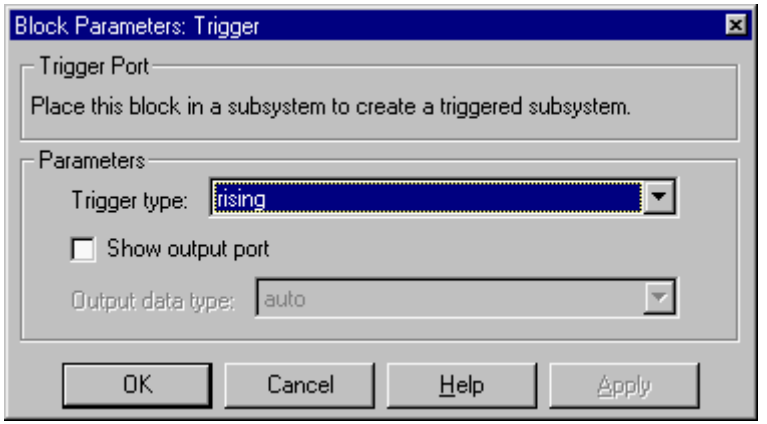
8.3 Exemple 2 : Compteur d'évènements

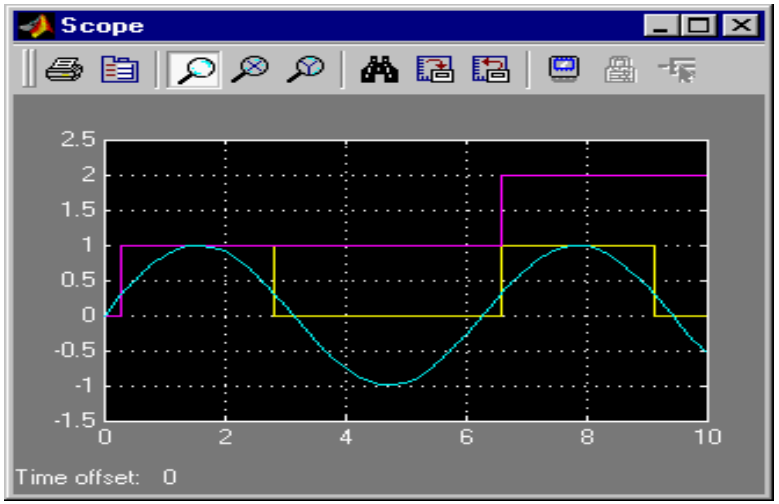
• **Le bloc *Unit Delay* : la condition initiale reste à 0. Le champ "*Sample Time*" est mis à -1 pour indiquer que ce bloc ne dépend pas du temps, puisque le sous-système est triggé.**



Le résultat de la simulation est :

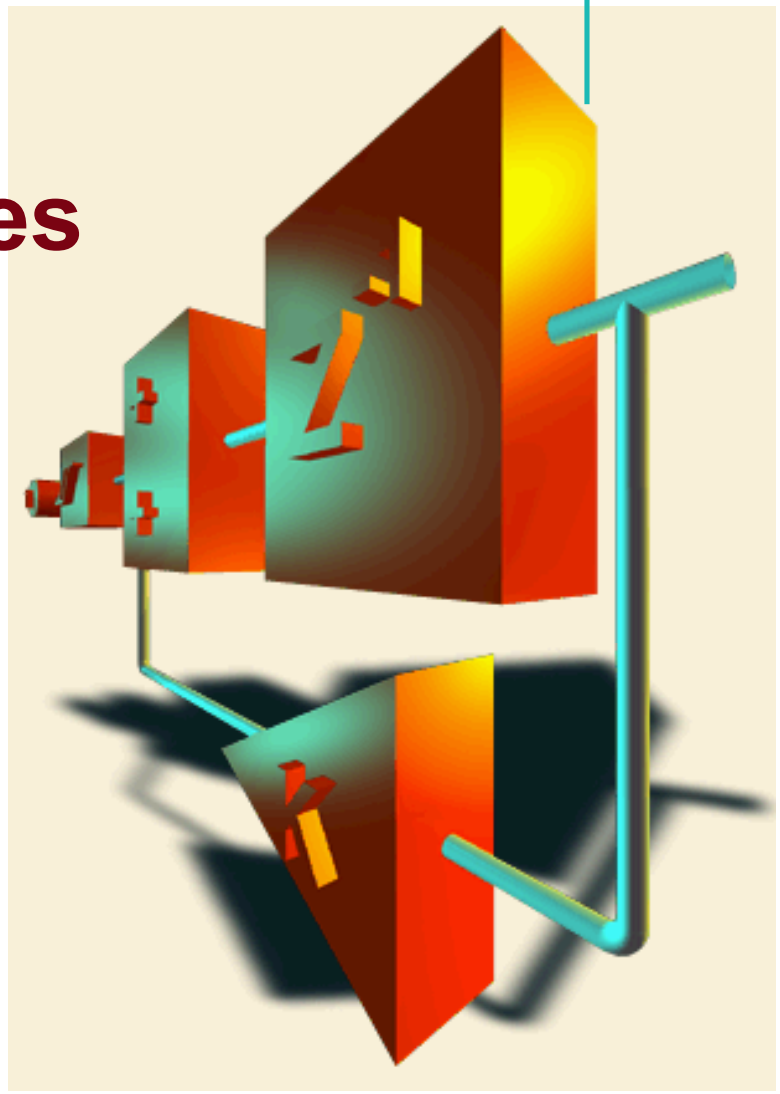
• **Le bloc *Trigger* : l'exécution est faite sur front montant.**





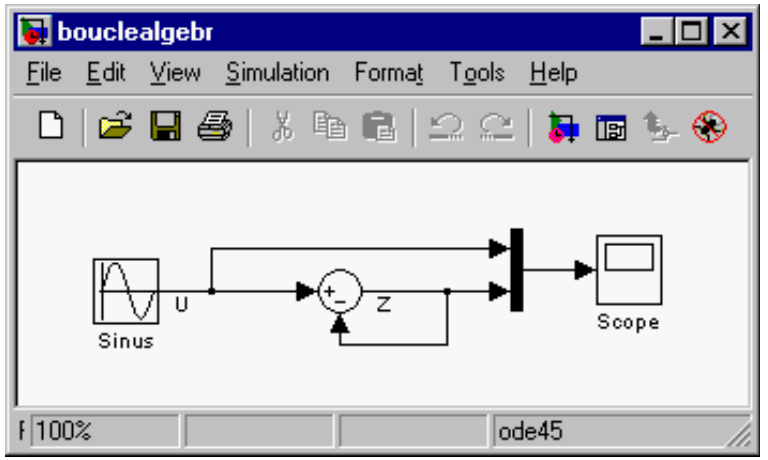
Simulink

9. Boucles Algébriques



• Certains blocs de SIMULINK ont la particularité d'avoir la sortie qui dépend *directement* de l'entrée. C'est le cas pour les blocs *Gain*, *Sum*, *Product*, ...

Dans certains cas, l'utilisation de ces blocs génère une **boucle algébrique**.

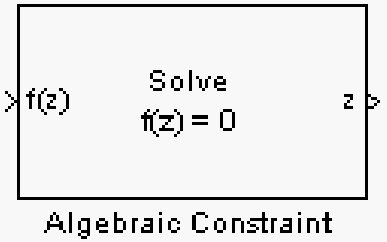


• La difficulté est de savoir quelle est la valeur de Z à un instant donné.

Mathématiquement, la solution est simple :

$$Z = U - Z \Leftrightarrow Z = U/2$$

• Parfois, la résolution de ce type de problème nécessite d'utiliser le bloc *Algebraic Constraint* de la bibliothèque *Math* pour contraindre l'algorithme de résolution.

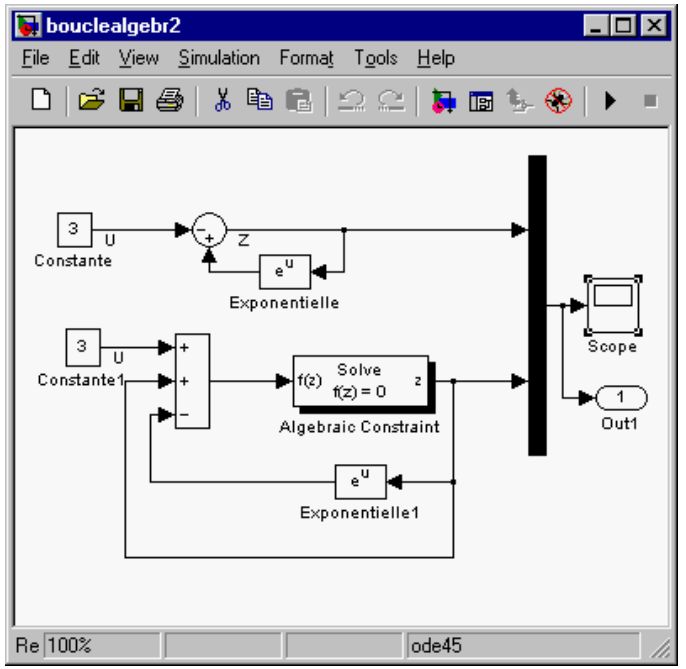


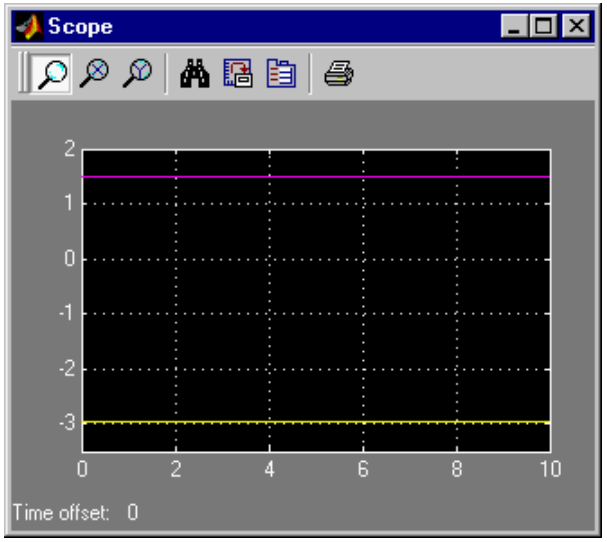
Exemple

Résoudre **f(z) = 0** avec **f(z) = 3 + z - exp(z)**

Cette équation a deux solutions. Avec le bloc ***Algebraic Constraint***, il est possible de donner une valeur initiale pour permettre à l'algorithme de se diriger vers une des solutions voulues.

Le modèle correspondant est présenté sous deux formes différentes, l'une utilisant le bloc ***Algebraic Constraint*** et l'autre non :





• La courbe mauve (courbe du haut) est la solution de l'équation donnée par la partie du modèle utilisant le bloc ***Algebraic Constraint*** . La courbe jaune (courbe du bas) est la solution de l'équation donnée par l'autre partie du modèle.

Exemple (cont.)

Ces deux valeurs sont réellement les solutions de l'équation f(z)=0 :

```
>> z=yout(end,:)
z =
   -2.9475    1.5052

>> Fz=3+z-exp(z)
Fz =
  1.0e-015 *
-0.1943         0
```

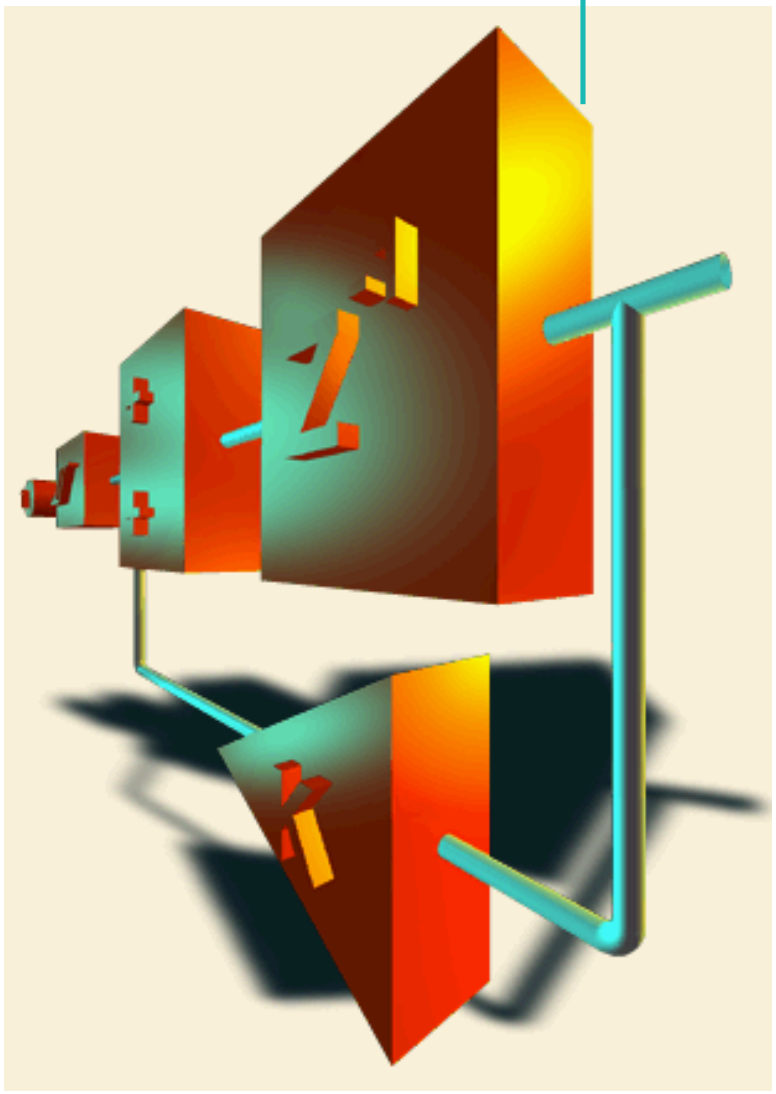
Exercice 2

Modifier le modèle pour que :

- **le compteur démarre de 0 et s'incrémente avec un pas de 2**
- **le compteur démarre de 10 et se décrémente avec un pas de 1**
- **le compteur démarre de 10, se décrémente avec un pas de 1, et la simulation s'arrête lorsque le compteur atteint 0**

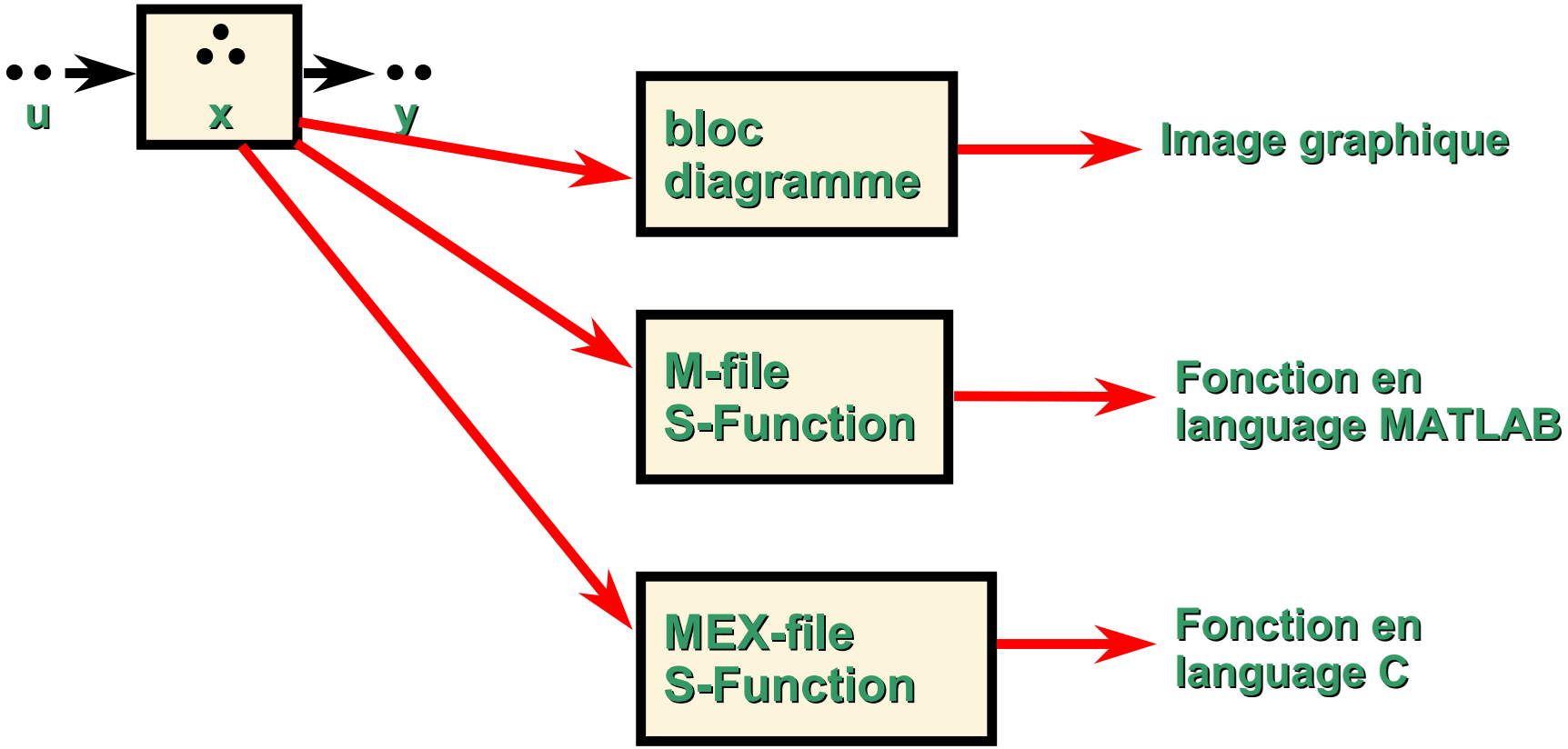
Simulink

10. S-Functions



10.1 Introduction

Une S-Function définit la dynamique d'un système. Il existe différent types de S-Functions :



10.1 Introduction (cont.)

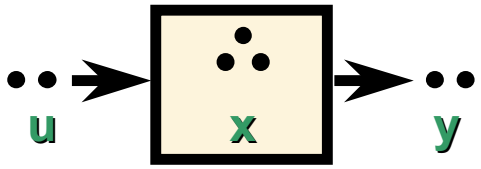
Les S-Functions servent à :

- **Créér de nouveaux blocs**
- **Incorporer des fonctions C dans SIMULINK**
- **Décrire des systèmes mathématiques tels que équations, boucles,...**
- **Incorporer du graphique MATLAB dans un modèle Simulink**

10.2 Comment travaillent les S-Functions

• **Une S-Function est réprésentée par :**

- **des entrées u**
- **des états x (continus, discrets, hybrides)**
- **des sorties y**



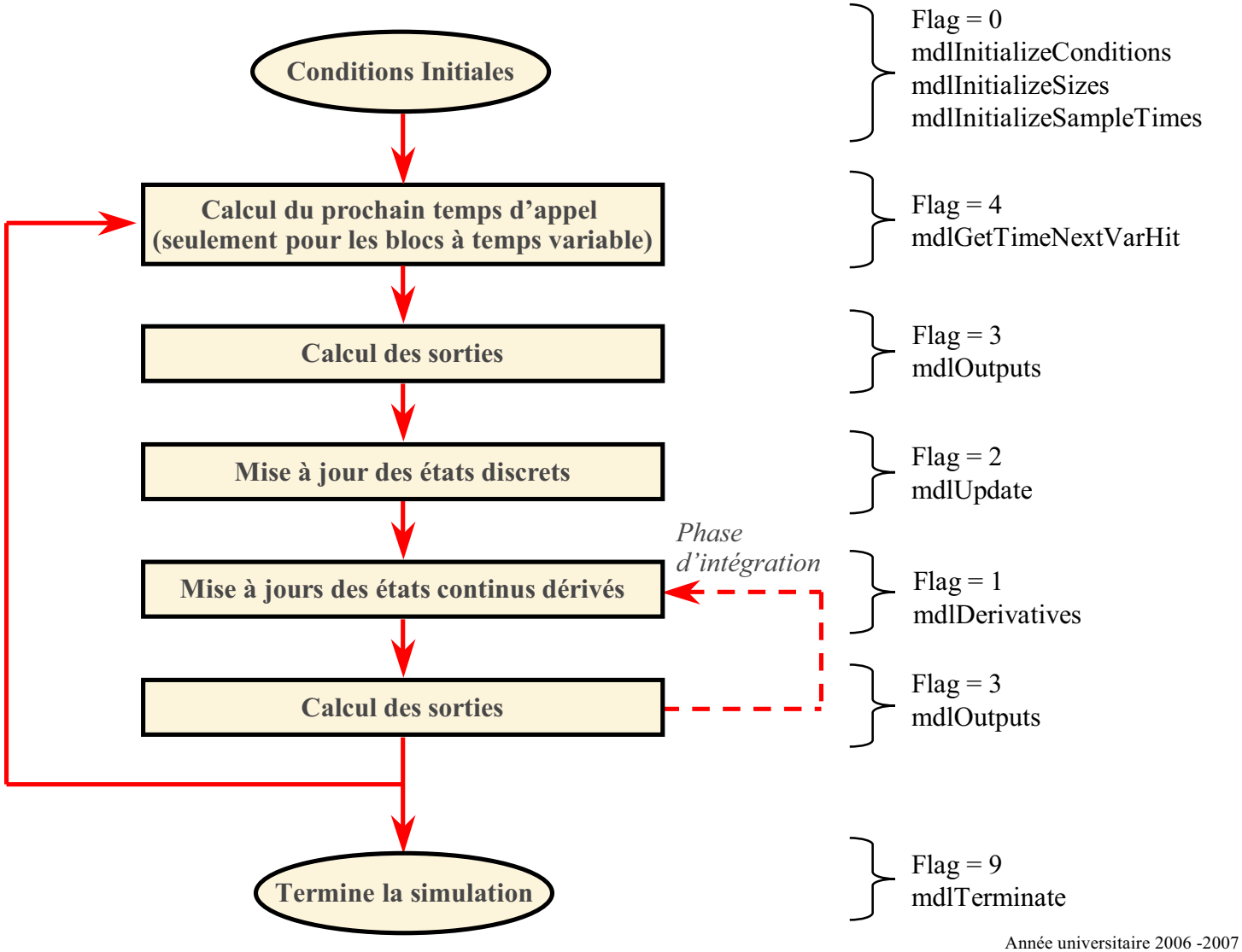
• **Durant la simulation, à chaque pas de calcul, SIMULINK appelle chaque bloc du modèle et calcule les sorties, met à jour les états discrets et intègre les états dérivés :**

- **Calcul des sorties :** $y = f(t,x,u)$
- **Mise à jour des états discrets :** $x_{d+1} = f_d(t,x,u)$
- **Mise à jour des états continus dérivés :** $\dot{x}_c = f_c(t,x,u)$

avec :

- x **: les états continus** x_c **et discret** x_d $\rightarrow$ $x = \begin{bmatrix} x_c \\ x_d \end{bmatrix}$
- u **: les entrées**
- t **: le temps** $\rightarrow$ $t = \begin{bmatrix} 0.0 & 0.0 \\ te_1 & Offset_1 \\ te_2 & Offset_2 \\ \cdots & \cdots \\ te_n & Offset_n \end{bmatrix}$

10.3 Séquencement d'une Simulation



10.4 Les S-Functions en M-File

- Les S-Functions en M-File ne sont pas optimales pour la vitesse ou la génération de code, mais sont utiles pour le prototypage rapide d'algorithme
- Utiliser le fichier template pour construire des S-Functions en langage MATLAB : `$matlab/toolbox/simulink/sfuntmpl.m`
- Utiliser la variable "flag" pour réagir aux différents appels
- Structurer la S-Function avec des instructions **switch/case** et des appels à des sous fonctions
- Déclarer la fonction avec les variables d'entrée/sortie dans cette ordre (obligatoire) :

```
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
%SFUNTMPL General M-file S-function template
%   With M-file S-functions, you can define you own ordinary differential
%   equations (ODEs), discrete system equations, and/or just about
%   any type of algorithm to be used within a Simulink block diagram.
```

```
>> edit sfuntmpl
```

10.4 Les S-Functions en M-File

La syntaxe est la suivante :

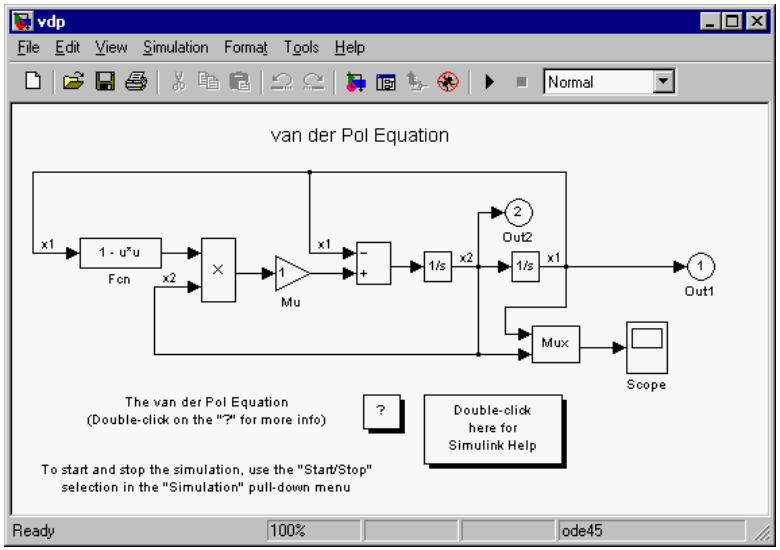
```
function [sys,x0,xstr,ts,xts] = modele(t,x,u,flag)
```

avec :

- Paramètres d'entrée :
  - t : temps courant de simulation,
  - x : vecteur d'état,
  - u : vecteur d'entrée,
  - flag : variable de contrôle de la simulation.
- Paramètres de sortie :
  - sys : variable de sortie dépendante de flag,
  - x0 : conditions initiales des états,
  - xstr : descriptif des états,
  - ts : périodes d'échantillonnage,
  - xts : périodes d'échantillonnage de chaque état.

10.5 Dimensionnement d'un Modèle

• **Pour déterminer les caractéristiques du modèle, on appelle la S-Function qui le représente à partir de MATLAB avec le `flag = 0`.**

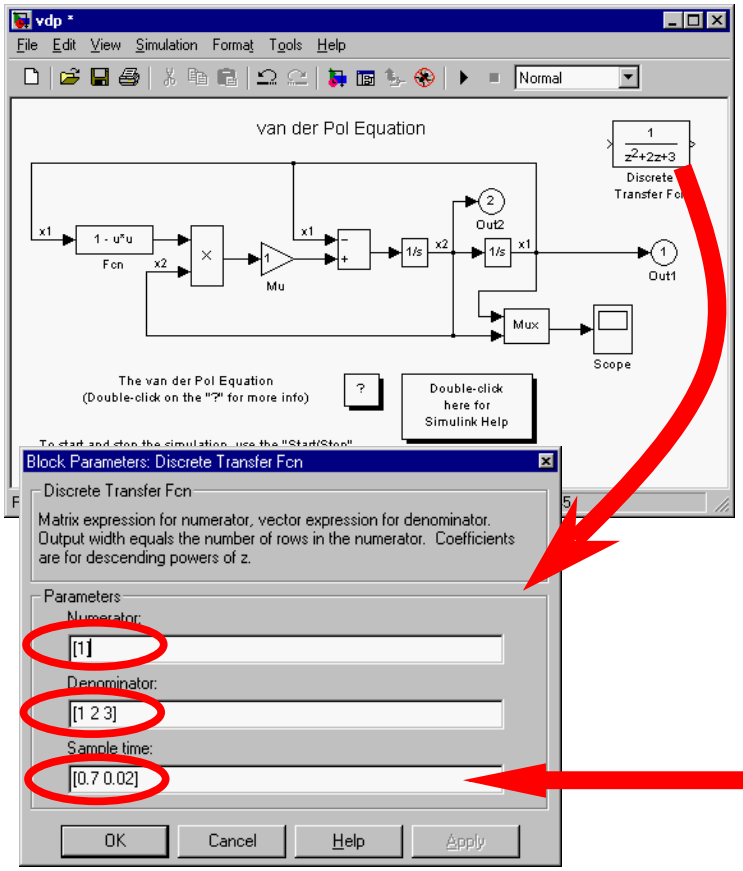


Exemple :

```
>> vdp
>> [sys,x0,xstr,ts]=vdp([],[],[],0)
sys =
     2          2 états continus
     0          0 états discrets
     2          2 sorties (2 blocs outport)
     0          0 entrée  (0 bloc inport)
     0          inutilisé
     0          pas de boucle algébrique
     1          une seule période d'échant.  égale à 0

x0 =
     2          conditions initiales de l'état integrator1
     0          conditions initiales de l'état integrator2

xstr =
     'vdp/x1'   le premier état s'appelle x1
     'vdp/x2'   le deuxième état s'appelle x2

ts =
     0    0     le système est continu
```

10.5 Dimensionnement d'un Modèle

• **En ajoutant un bloc discret, la S-Function graphique `vdp` devient un système hybride avec mélange d'états continus et discrets.**




```
>> [sys,x0,xstr,ts]=vdp([],[],[],0)
sys =
     2          2 états continus
     2          2 états discrets (Transfert Fcn d 'ordre 2)
     2          2 sorties (2 blocs outport)
     0          0 entrée  (0 bloc inport)
     0          inutilisé
     0          pas de boucle algébrique
     2          2 périodes d'échant.  égales à 0 et à 0.7

x0 =
     2          conditions initiales de l 'état integrator1
     0          conditions initiales de l 'état integrator2
     0          } Conditions initiales de la fonction de
     0          } transfert discrète d 'ordre 2

xstr =
    'vdp/x1'    le premier état s'appelle x1
    'vdp/x2'    le deuxième état s'appelle x2
    'vdp/Discrete Transfert Fcn'
    'vdp/Discrete Transfert Fcn'

ts =
    0      0          partie continue
    0.700  0.0200     partie discrète
```

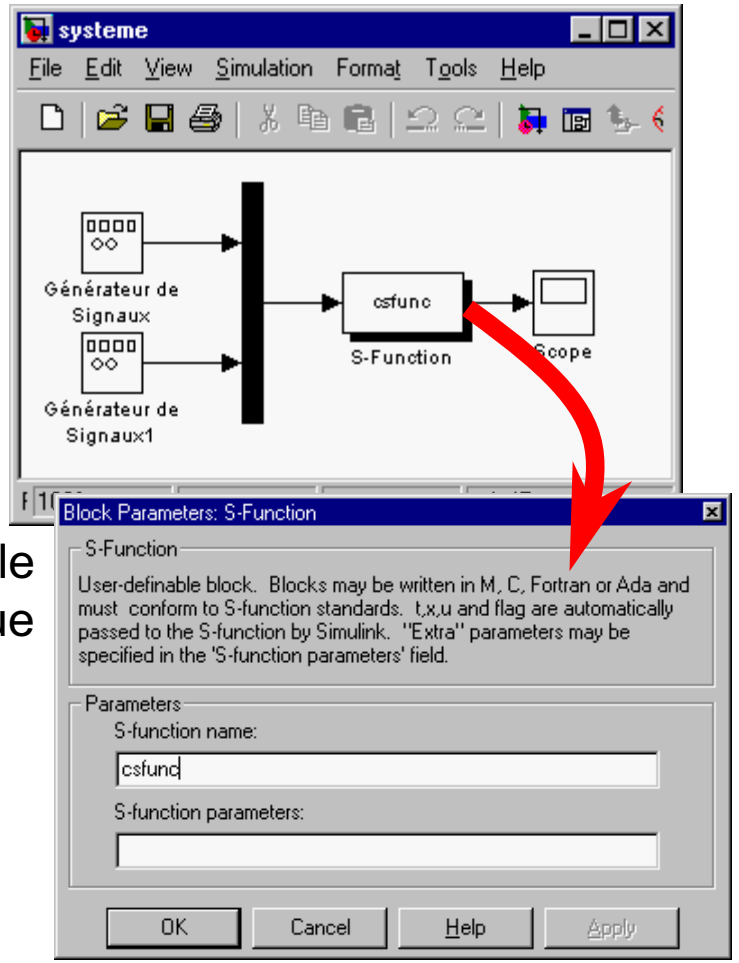
Echantillonnage du bloc tous les : `n*Te + offset`

10.6 S-Functions en langage MATLAB

- **Modèle continu**

Soit le système à 2 entrées, 2 sorties et 2 états, décrit par :

$$\begin{cases} \dot{x} = \begin{bmatrix} -0.09 & -0.01 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 & -7 \\ 0 & -2 \end{bmatrix} u \\ y = \begin{bmatrix} 0 & 2 \\ 1 & -5 \end{bmatrix} x + \begin{bmatrix} -3 & 0 \\ 1 & 0 \end{bmatrix} u \end{cases}$$



La S-Function est appelée dans le modèle par le bloc *S-Function* de la bibliothèque *Functions & Tables* de SIMULINK.

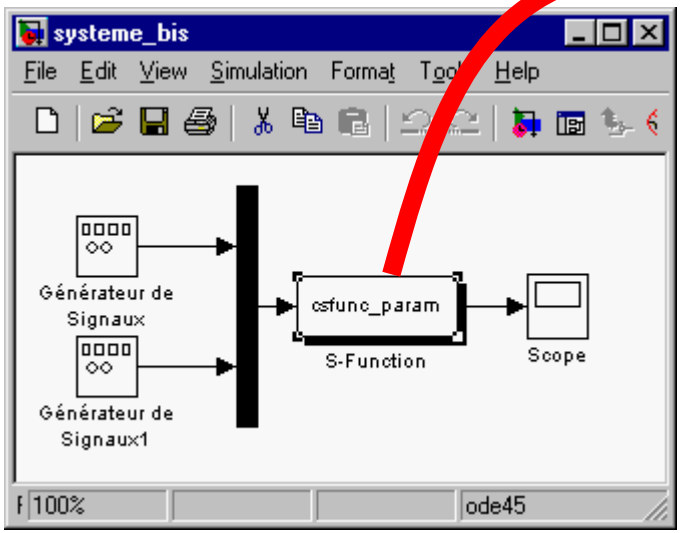
Pour éditer la S-Function, taper :

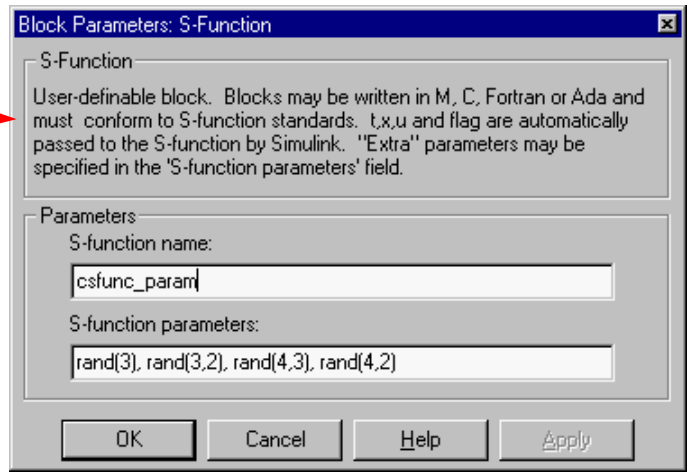
```
>> edit csfunc.m
```

10.6 S-Functions en langage MATLAB

- **Modèle continu avec passage de paramètres**

  • Le modèle sous forme d'espace d'états est repris, mais cette fois-ci les quatres matrices A, B, C, D sont définies à l'extérieur de la S-Function. Ces quatres matrices sont des *paramètres*.





• Pour éditer la S-Function, taper :

```
>> edit csfunc_param.m
```

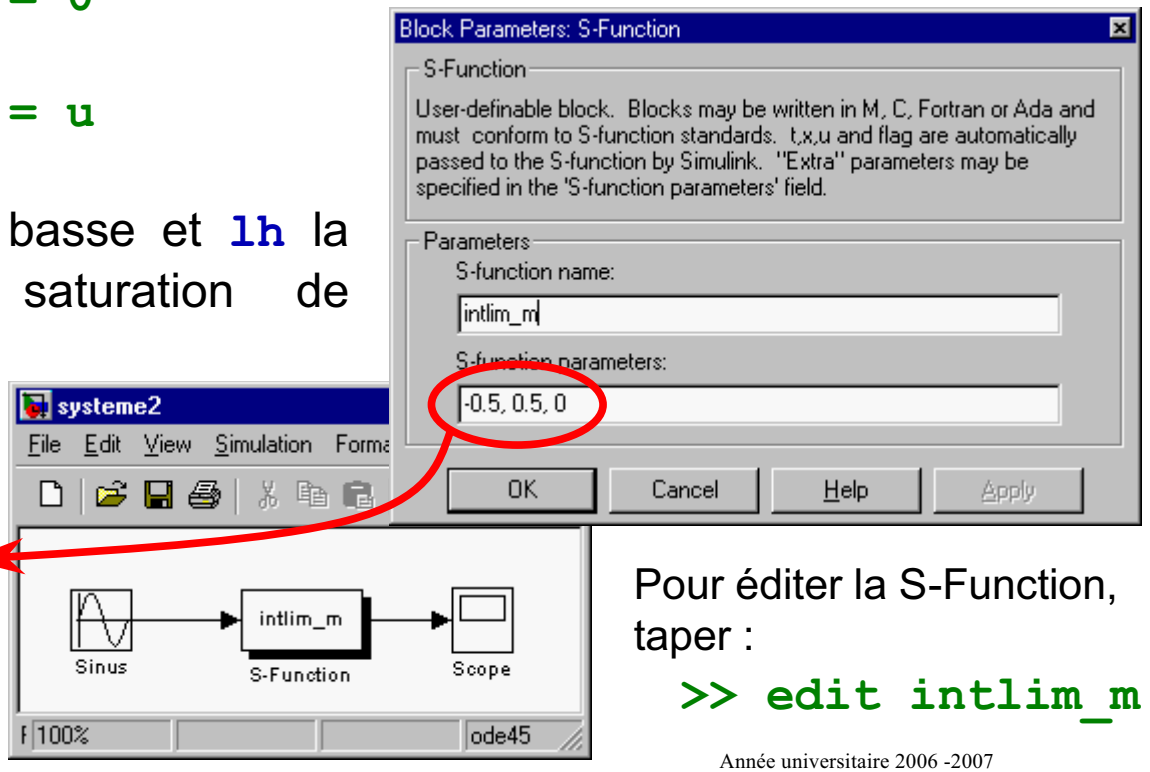
10.6 S-Functions en langage MATLAB

- **Modèle continu : intégrateur limité (saturé)**

Le modèle est exprimé sous forme d'une équation différentielle du 1er ordre :

```
si (x <= lb et u < 0) ou (x >= lh et u > 0) alors
        x' = 0
sinon
        x' = u
```

où **lb** est la limite basse et **lh** la limite haute de saturation de l'intégration.



les paramètres de la S-Function lb, lh et xi sont définis dans le champ " *Function parameters* ". Dans le cas présent, on a :

lb=-0.5 lh=0.5 xi=0

Pour éditer la S-Function, taper :

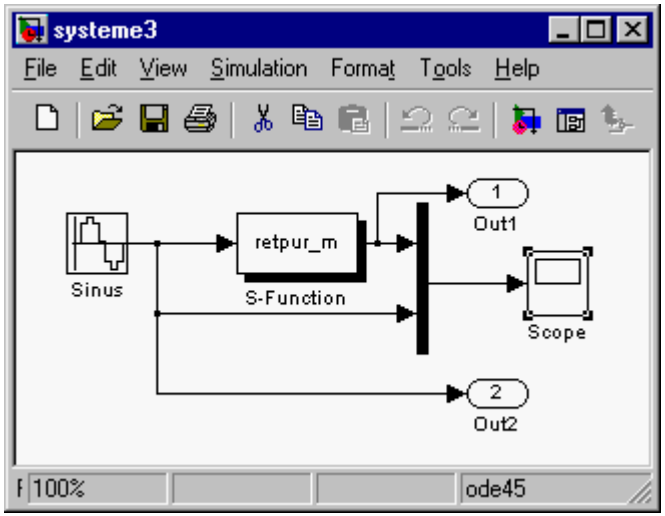
```
>> edit intlim_m
```

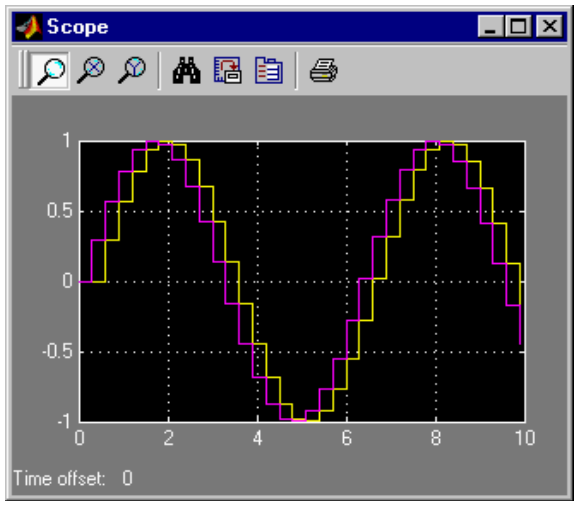
10.6 S-Functions en langage MATLAB

- **Modèle discret : retard pur**

• Cette S-Function exemple applique un retard pur de 0.3 s sur le signal d 'entrée. Pour éditer la S-Function, taper :

```
>> edit retpur_m
```





• La courbe mauve correspond au sinus d'entrée, la courbe jaune au sinus retardé de 0.3 s.

• Le scope détecte le fait que les deux signaux sont discrets : l'affichage est fait en escalier.

10.7 S-Functions en langage C

- **Introduction**
  - **Les S-Functions en langage C sont optimales pour la vitesse ou la génération de code**
  - **Utiliser le fichier template pour construire des S-Functions en langage C :**

    `matlabroot\simulink\src\sfuntmpl.c`
  - **Utiliser des macros pour accéder aux paramètres optionnels et aux signaux de sortie**
  - **Un compilateur ANSI C est nécessaire**
    - **Configurer la commande mex en tapant :**

      `>> mex -setup`
  - **Compiler la S-Function C en invoquant le compilateur C par la commande mex :**

    `>> mex ma_fonction_C.c`

10.7 S-Functions en langage C

- **Fonctions propres à la S-Function**

• Initialisation	- `mdlInitializeSizes` - `mdlInitializeSampletimes` - `mdlInitializeConditions`
• Calcul des sorties	- `mdlOutputs`
• Mise à jour des états discrets	- `mdlUpdate`
• Calcul du prochain temps d'appel (option)	- `mdlGetTimeOfNextVarHit`
• Mise à jour des états continus dérivés	- `mdlDerivatives`
• Tâches de fin de simulation	- `mdlTerminate`

10.7 S-Functions en langage C

- **Macros d ’accès à la structure Simstruct (initialisation)**

Champ de la Structure	Macro pour appliquer une valeur
Nombre d'états continus	**ssSetNumContStates(S, numContStates)**
Nombre d'états discrets	**ssSetNumDiscStates(S, numDiscStates)**
Nombre de sorties	**ssSetNumOutputs(S, numOutputs)**
Nombre d'entrées	**ssSetNumInputs(S, numInputs)**
Flag pour boucle algébrique	**ssSetDirectFeedthrough(S, dirFeedThru)**
Nombre de périodes d'échantillonnage	**ssSetNumSampleTimes(S, numSamplesTimes)**
Nombre d'arguments d'entrée	**ssSetNumInputArgs(S, numInputArgs)**
Nombre de vecteurs de travail de type entier	**ssSetNumIWork(S,numIWork)**
Nombre de vecteurs de travail de type réel	**ssSetNumRWork(S,numIWork)**
Nombre de vecteurs de travail de type pointeur	**ssSetNumPWork(S,numIWork)**

- **Il existe également des macros pour lire ces valeurs (préfix ssGet)**

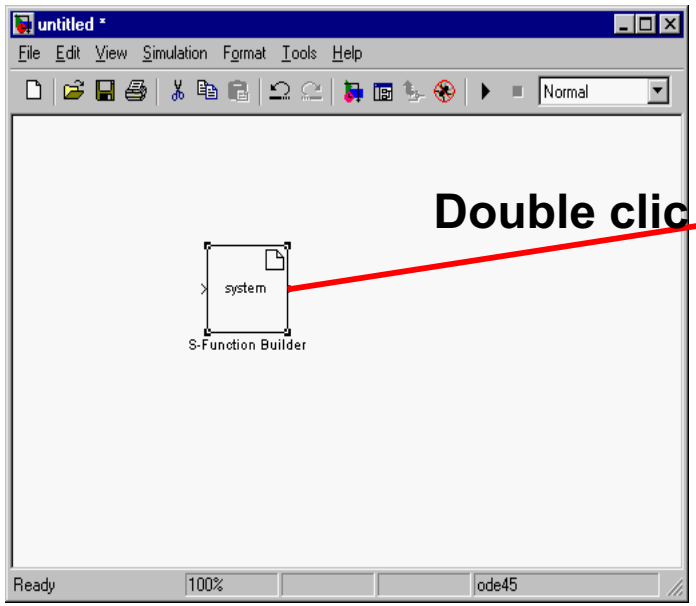
10.7.1 Constructeur de S-Function

• **Le constructeur de S-Function (*S-Function Builder*) est un bloc Simulink qui construit une S-Function à partir des caractéristiques et du code de C fournis par l'utilisateur.**

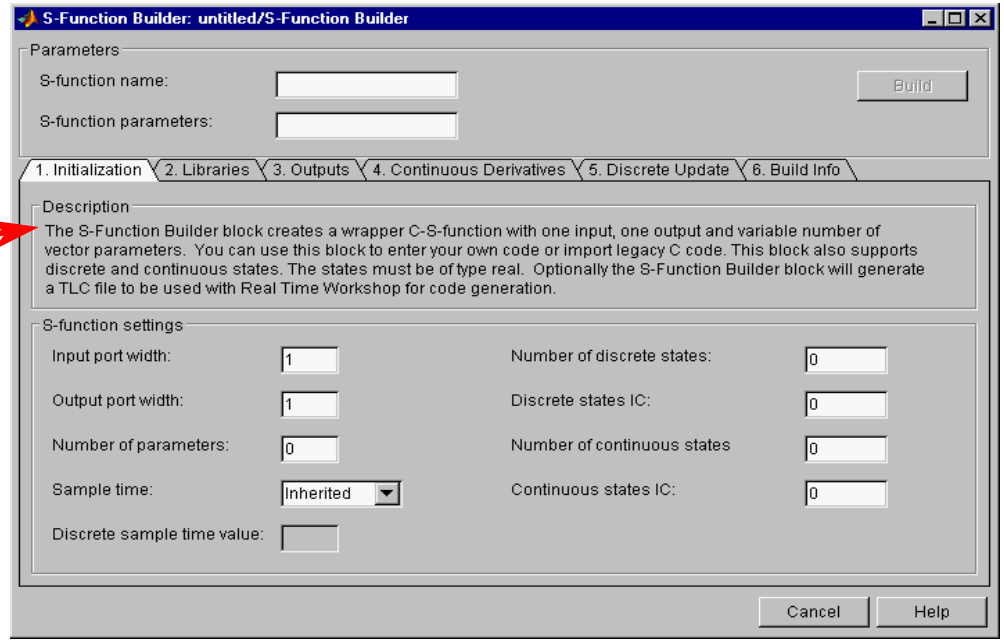
• **Pour construire une S-Function avec le S-Function Builder :**

1. Basculez du répertoire MATLAB courant au répertoire dans lequel vous voulez créer la S-Function.

2. Placez une copie du bloc S-Function Builder de la bibliothèque « Functions&Tables » dans le modèle simulink.



Double clic

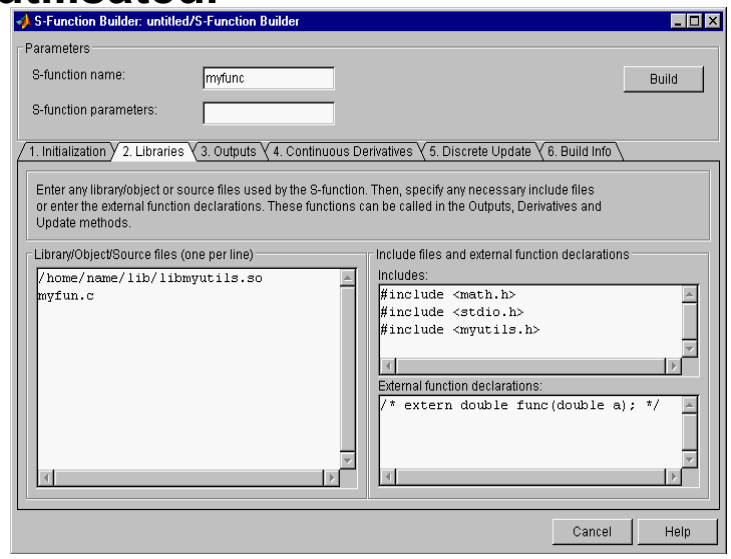


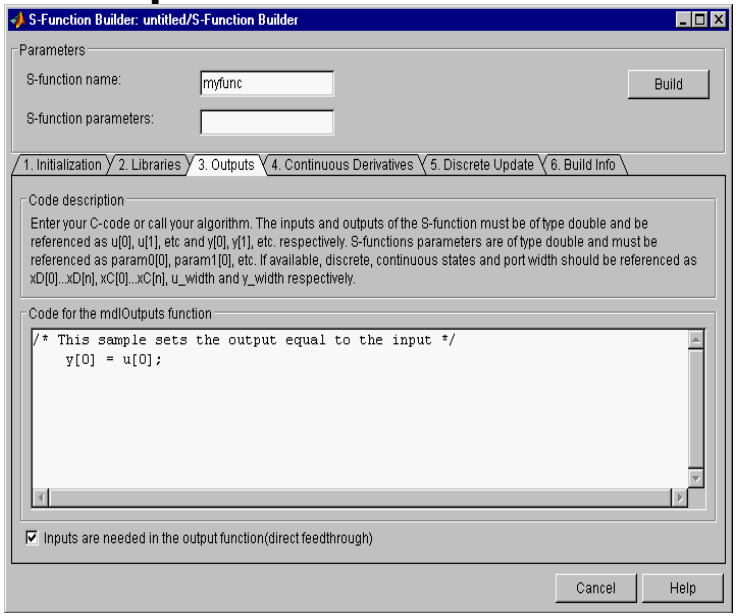
10.7.1 Constructeur de S-Function

- **Ecrivez le nom de la S-Function dans le champs *S-function name*.**

- **Si la S-Function a des paramètres, écrivez les valeurs par défaut pour les paramètres dans le champ *S-Function parameters*.**

- **Utilisez les panneaux de spécification et du code sur la boîte de dialogue du bloc « S-Function Builder » pour inscrire le code source utilisateur**





- **Après configuration du mex (mex –setup) cliquez sur Build. Simulink construit le MEX-File qui met en application la S-Function et sauvegarde le fichier dans le répertoire courant.**

10.7 S-Functions en langage C

- **Exemple**

On reprend le modèle continu sous forme d'espace d'état :

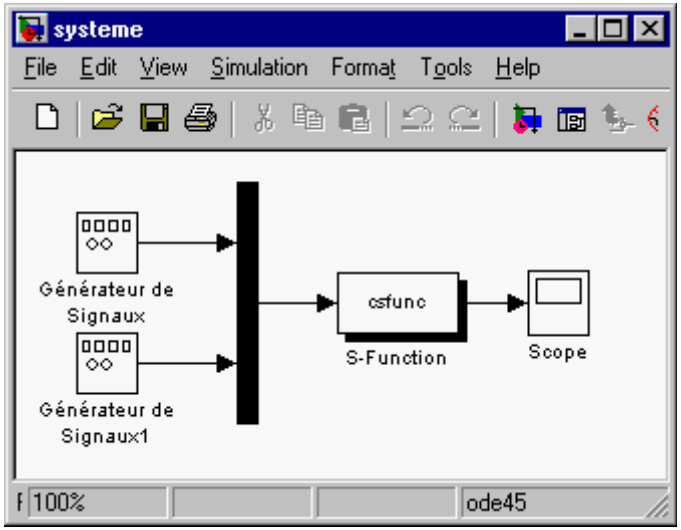
```
>> edit csfunc.c
```

La compilation se fait par la commande :

```
>> mex csfunc.c
```

qui crée le fichier csfunc.dll.

Cette S-Function est appelée dans SIMULINK avec le bloc *S-Function*.

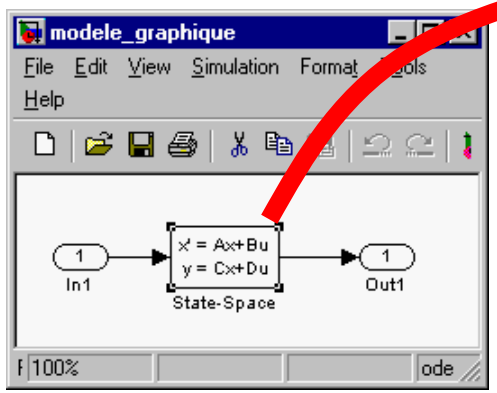


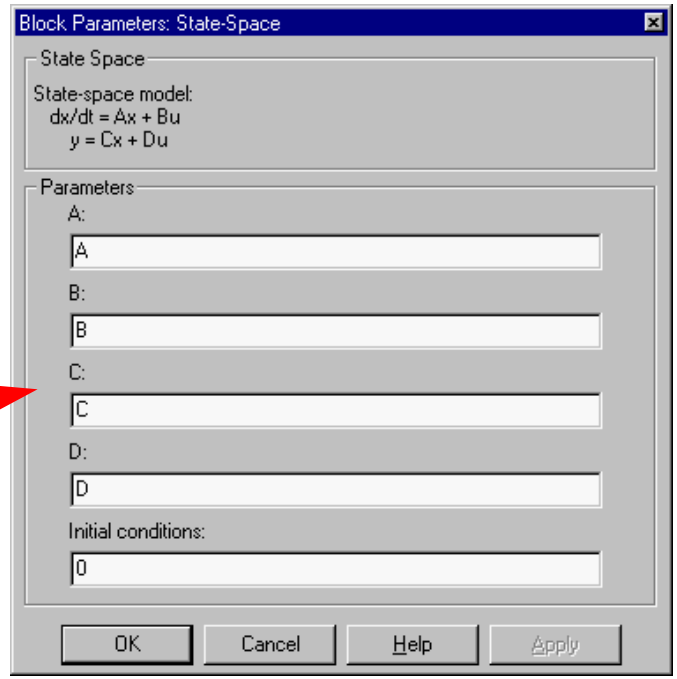
10.8 Comparaison des différents types de représentation

• Les trois types de représentation (bloc graphique, S-Function en langage MATLAB, S-Function en langage C) sont appelés de MATLAB en mesurant le temps d'exécution.

L'exemple utilisé est un modèle sous forme d'espace d'état défini par A, B, C, D.

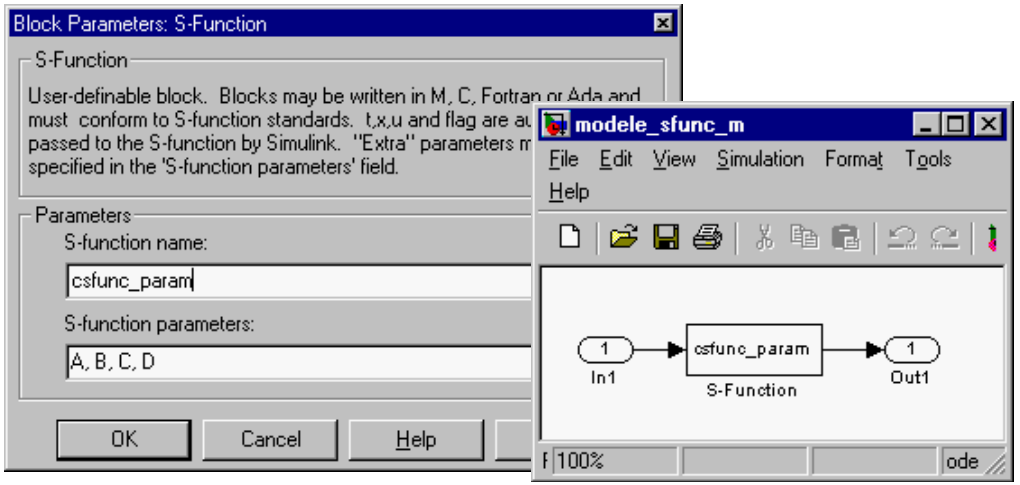
Le modèle *modele_graphique* correspond à la représentation sous forme de bloc graphique



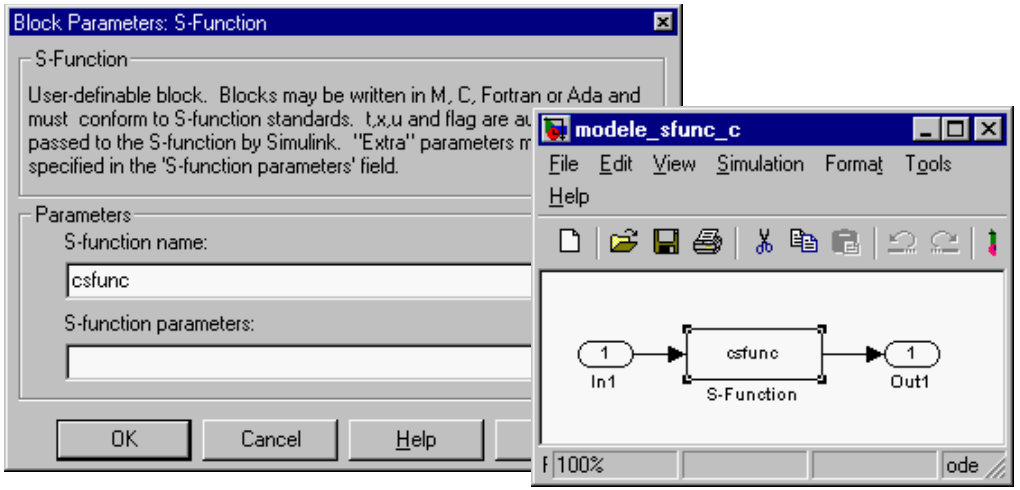


10.8 Comparaison des différents types de représentation (cont.)

• Le modèle *modele_sfunc_m* correspond à la S-Function en langage MATLAB :



• Le modèle *modele_sfunc_c* correspond à la S-Function en langage C :



10.8 Comparaison des différents types de représentation (cont.)

```
% Comparaisons des temps de simulation
% des différentes représentation :
%          . modèle graphique
%          . S-Function en langage Matlab
%          . S-Function en langage C

clear all

A = [-0.09   -0.01
      1           0];
B = [ 1       -7];
C = [ 0     2
      1    -5];
D = [-3     0
      1     0];

options = simget('modele_graphique');
...
```

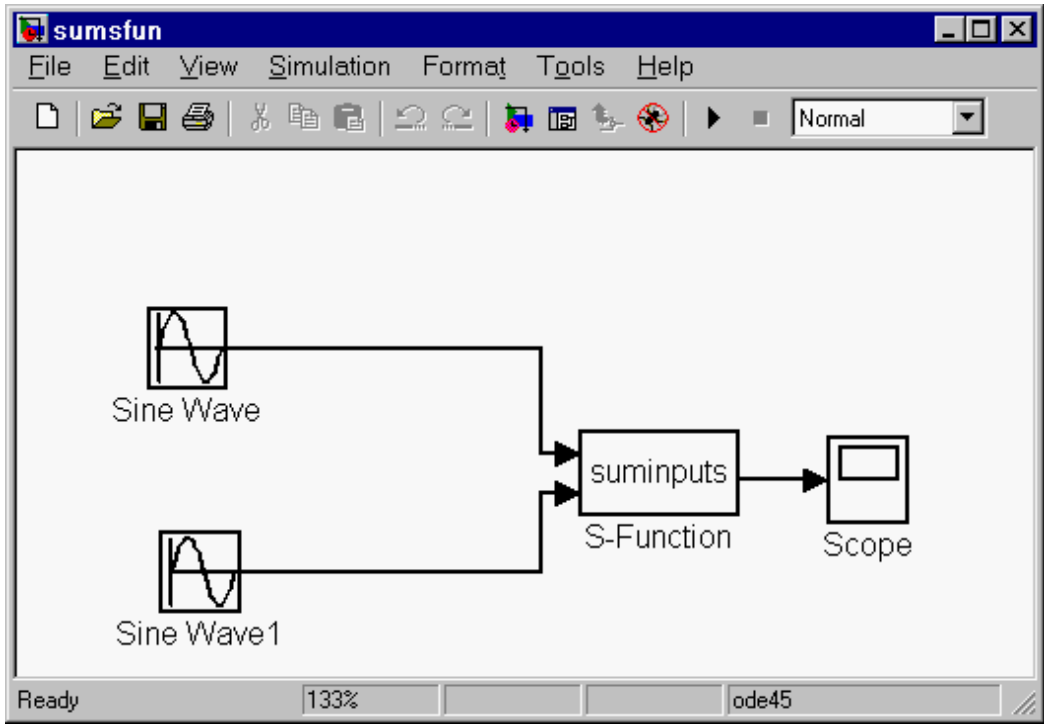
```
...
ListeModele = {…
 'modele_graphique',…
 'modele_sfunc_m',…
 'modele_sfunc_c'};

for i=1:3
   tic
   [t,x,y] = sim(…
      ListeModele{i},…
      [0:0.1:200],…
      options,…
      '[sin(t),cos(t)]');
   t = toc;
   disp(ListeModele{i})
   disp(t)
end
```

Pour lancer ce script, taper :

```
>> compare_modele
```

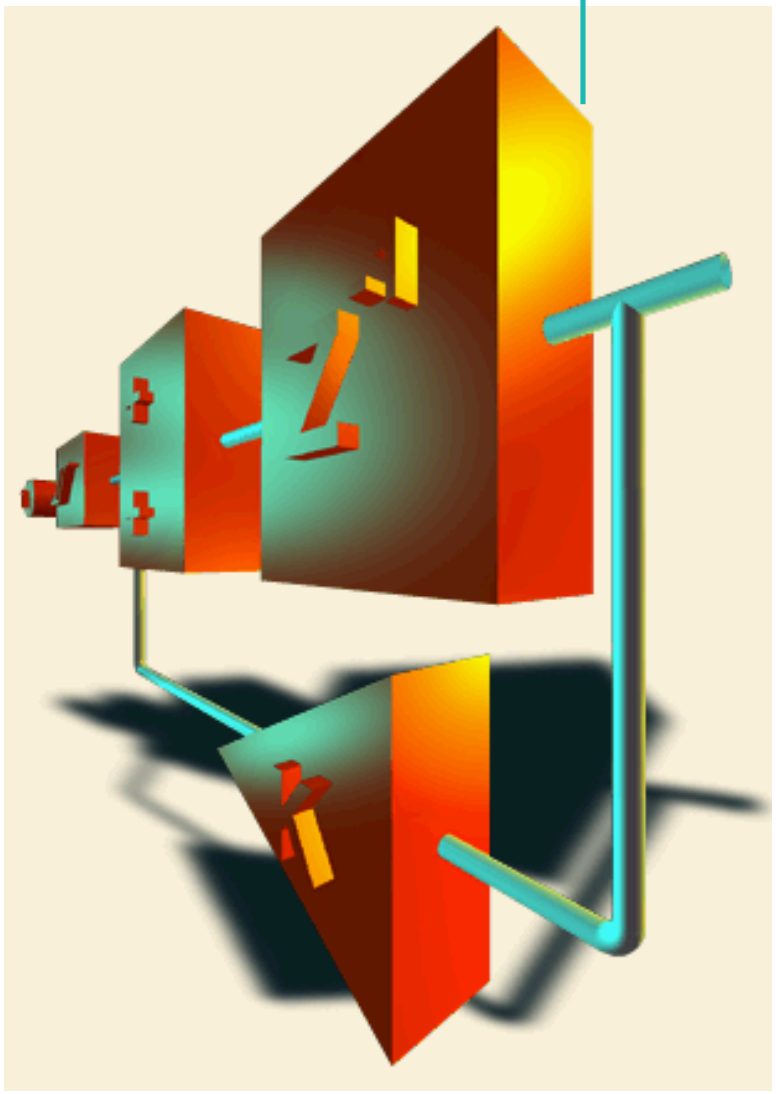
10.9 Exercice

- **Écrivez une S-Function pour créer un bloc ayant deux ports d'entrées et un port de sortie. La largeur des ports d'entrée et du port de sortie est identique.**
- **Le vecteur de sortie est la somme des deux vecteurs d'entrées**



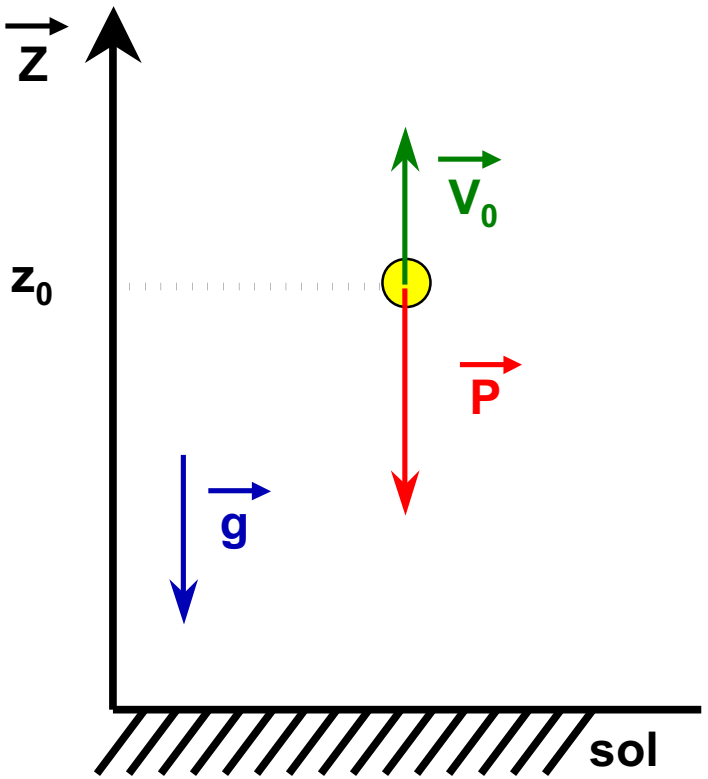
Simulink

Exercice :
Rebond d'une Balle



L'objectif de cet exercice est d 'utiliser toutes les particularités du bloc Integrator de la bibliothèque Continuous.

- Modéliser la chute libre d'une balle sur un sol, sachant que la hauteur initiale est de 10 m, la vitesse initiale est de 5m/s vers le haut et l'amortissement du sol est de 0.8
- Compléter le modèle de manière à ce que la simulation s'arrête lorsque l'amplitude des rebonds est "faible" (par exemple, amplitude inférieure à 5 cm)
- Compléter le modèle pour représenter une balle qui se déplace dans un plan vertical avec une vitesse initiale colinéaire au sol de 0.1 m/s, entre les abscisses 0 m et 5 m. Utiliser le bloc *XY Graph* pour l'affichage. Les murs sont considérés sans amortissement
- Compléter le modèle pour représenter une balle qui se déplace dans un cube dont les quatre murs sont placés à 5 m l'un de l'autre. La balle est placée à l'origine à une hauteur de 10 m. Les murs sont considérés sans amortissement. Puis, créer une S-Function en langage MATLAB pour l'affichage



- **Relation Fondamentale de la Dynamique :**

$$m\vec{\Gamma} = \sum \vec{F}_{ext}$$

- **Données Numériques :**
  - $z_0 = 10$ m
  - $V_0 = 5$ m.s^{-1}